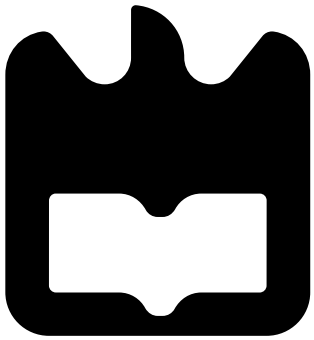
Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2012

João Gilberto Reigota Afonso

Plataforma de Gestão de Redes Veiculares



João Gilberto Reigota Afonso

Plataforma de Gestão de Redes Veiculares

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica da Dra. Susana Sargento, Professora Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Prof. Dr. João Nuno Pimentel da Silva Matos
Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro (por delegação do Reitor da Universidade de Aveiro)

vogais / examiners committee

Prof. Dra. Susana Isabel Barreto de Miranda Sargento
Professora Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro (Orientadora)

Prof. Dr. Daniel Enrique Lucani Rotter
Professor Auxiliar Convidado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto (Arguente Principal)

agradecimentos / acknowledgements

Agradeço aos meus pais, por todo o apoio que me deram ao longo destes anos, e por me terem dado a oportuninade e os meios para fazer este curso. À minha namorada, também um grande obrigado, por toda a ajuda, compreensão e dedicação durante estes longos anos de estudo. Dou também os meus agradecimentos aos meus amigos que estiveram comigo durante estes anos, por todos os momentos e pelo apoio que proporcionaram.
Devo também os meus agradecimentos aos colaboradores do grupo de trabalho do DRIVE-IN de Aveiro, por toda a ajuda e apoio prestados durante a realização deste trabalho. Agradeço à Prof. Dra. Susana Sargento pela orientação e apoio que me prestou no decorrer deste trabalho, sem os quais não teria sido possível tê-lo realizado.

Palavras-chave DRIVE-IN, VANET, IEEE 802.11p, *testbed*, Sistemas de gestão de *testbed*, OMF

Resumo

O *Distributed Routing and Infotainment through VEhicular Inter - Networking* (DRIVE-IN) surge como um projeto na área das *Vehicular Ad-hoc NETwork* (VANET), com o intuito de investigar como a comunicação entre veículos pode melhorar a experiência dos utilizadores e a eficiência em geral da utilização de veículos e estradas. Tem vindo a ser desenvolvida, no contexto deste projeto, uma *tesbed* que consiste em 465 táxis na cidade do Porto, utilizando a norma IEEE 802.11p, criada especificamente para as redes veiculares. Com a criação desta *testbed* surgem novos desafios: a gestão de todos os seus recursos, bem como a execução de experiências e recolha de resultados das mesmas. Este trabalho visa responder a este problema com a implementação de um sistema de gestão da *testbed*, utilizando para isso o *cOntrol and Management Framework* (OMF). Esta implementação requereu algumas alterações ao funcionamento base do OMF, como a adaptação ao funcionamento com a rede celular ou a geração de novas imagens de disco e a sua integração com a plataforma de gestão. Após realizada a implementação obtiveram-se resultados de *upload* de *software* e de experiências para os nós da *testbed*, e comunicação entre eles usando a interface *wireless* da norma IEEE 802.11p, assim como a recolha dos resultado das experiências e de informação de gestão dos nós veiculares. Os resultados obtidos mostram que é possível executar múltiplas experiências que envolvem a utilização da VANET, recolhendo os dados resultantes. Estes resultados permitem concluir que esta plataforma de gestão tem bom desempenho, sendo viável para implementação numa rede deste tipo.

Keywords DRIVE-IN, VANET, IEEE 802.11p, testbed, testbed management system, OMF

Abstract Distributed Routing and Infotainment through VEhicular Inter - Networking (DRIVE-IN) is a project in the area of Vehicular Ad-hoc NETwork (VANET), in order to investigate how the communication between vehicles can improve the user experience and the overall efficiency in the use of vehicles and roads. A tesbed has been developed in the context of this project, consisting of 465 taxis in Porto, using the standard IEEE 802.11p, created specifically for vehicular networks. New challenges emerge from the creation of this testbed: the management of all its resources, as well as the conduct of experiments and collection of its results. This work aims to answer this problem by implementing a testbed management system, using the cOntrol and Management Framework (OMF). This implementation required some changes to the base functionalities of OMF, as the adaptation to work with the cellular network or the generation of new disk images and their integration with the management platform. After performing the implementation, results were obtained with the upload of software and experiments to the testbed nodes and communication between them, using the wireless IEEE 802.11p interface, as well as the collection of results from experiments and the management information of the vehicular nodes. The results show that it is possible to run multiple experiments involving the use of VANET, collecting the resulting data. These results allow us to conclude that this management platform performs well and is feasible for implementation in this type of network.

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

AD *Application Definition*

AM *Aggregate Manager*

AP *Access-Point*

API *Application Programming Interface*

ASV *Advanced Safety Vehicle*

AU *Application Unit*

BER *Bit Error Rate*

bps *bits per second*

C2C *Car-to-Car*

C2C-CC *Car-to-Car Communication Consortium*

CALM *Continuous Air Interface for Long and Medium distances*

CAMP *Collision Avoidance Metrics Partnership*

CCA *Cooperative Collision Avoidance*

CPU *Central Processing Unit*

CRN *Congested Road Notification*

DBMS *DataBase Management System*

DES-TBMS *Distributed Embedded Systems Testbed Management System*

DHCP *Dynamic Host Configuration Protocol*

DRIVE-IN *Distributed Routing and Infotainment through VEhicular Inter - Networking*

DSRC *Dedicated Short-Range Communications*

EC *Experiment Controller*

ED *Experiment Description*

ETSI *European Telecommunications Standards Institute*

EWM *Emergency Warning Message*

FCC *Federal Communication Commission*

FCT Fundação para a Ciência e Tecnologia

GPS *Global Positioning System*

GPRS *General Packet Radio Service*

GSM *Global System for Mobile communications*

HRN *Human Readable Name*

HTTP *Hypertext Transfer Protocol*

ICMP *Internet Control Message Protocol*

IEEE *Institute of Electrical and Electronics Engineers*

IETF *Internet Engineering Task Force*

IMTT Instituto da Mobilidade e dos Transportes Terrestres

IP *Internet Protocol*

IPv6 *Internet Protocol version 6*

ISO *International Organization for Standardization*

IT Instituto de Telecomunicações

ITS *Intelligent Transportation System*

JAMES *JAva test-bed ManagEment System*

LAN *Local Area Network*

MAC *Medium Access Control*

MANET *Mobile Ad hoc Network*

NAT *Network Address Translation*

MCS *Measurement Collection Service*

ML *Measurement Library*

MP *Measurement Point*

NICTA *National ICT Australia*

OBU *On Board Unit*

OEDL *OMF Experiment Description Language*

OMF *cOntrol and Management Framework*

OML *OMF Measurement Library*

OSI *Open Systems Interconnection*

PC *Personal Computer*

PCN *Post Crash Notification*

PDA *Personal Digital Assistant*

PIN *Personal Identification Number*

PPP *Point-to-Point Protocol*

PXE *Preboot eXecution Environment*

RAM *Random-Access Memory*

RC *Resource Controllers*

RHCN *Road Hazard Control Notification*

RSU *Road Side Unit*

SNA *Sensor Network Athentication*

SNMP *Simple Network Management Protocol*

SOAP *Simple Object Access Protocol*

SSH *Secure Shell*

SSO *Single-Sign On*

STS *See-Through System*

SVA *Slow/Stop Vehicle Advisor*

TCP *Transmission Control Protocol*

TFTP *Trivial File Transfer Protocol*

TMN Telecomunicações Móveis Nacionais, S.A

UDP *User Datagram Protocol*

UMTS *Universal Mobile Telecommunications System*

URI *Uniform Resource Identifier*

USB *Universal Serial Bus*

V2I *Vehicle-to-Infrastructure*

V2R *Vehicle-to-Road*

V2V *Vehicle-to-Vehicle*

VANET *Vehicular Ad-hoc NETwork*

VIIC *Vehicle Infrastructure Integration Consortium*

VLAN *Virtual Local Area Network*

VSC *Vehicle Safety Consortium*

WAVE *Wireless Access in Vehicular Environments*

WBSS *WAVE mode Basic Service Set*

WiMAX *Worldwide Interoperability for Microwave Access*

WLAN *Wireless Local Area Network*

WSA *WAVE Service Advertisement*

WSDL *Web Services Description Language*

WSMP *WAVE Short Message Protocol*

WSN *Wireless Sensor Network*

XML *Extensible Markup Language*

XMPP *eXtensible Messaging and Presence Protocol*

ZOR *Zone of Relevance*

Capítulo 1

Introdução

Este documento foi realizado no âmbito da disciplina de Dissertação do Mestrado Integrado em Engenharia de Computadores e Telemática, da Universidade de Aveiro, com o tema de "Plataforma de Gestão de Redes Veiculares". Esta Dissertação está enquadrada no projecto *Distributed Routing and Infotainment through VEhicular Inter - Networking* (DRIVE-IN) [1].

Neste capítulo, são apresentadas as motivações, objetivos e enquadramento deste trabalho.

1.1 Motivação

À medida que os sistemas de GPS, tecnologias de sensores e interfaces *wireless* se tornam comodidades comuns, espera-se que, num futuro próximo, muitos dos veículos nas estradas sejam capazes de comunicar entre si, incluindo automóveis particulares, veículos comerciais, como camiões de transporte de mercadorias, e veículos de utilidade pública, como táxis ou autocarros.

A comunicação entre os veículos tem por base vários objetivos, dentro dos quais está a transmissão de informações de trânsito, como congestionamentos, e de avisos de emergência, como em caso de acidentes ou travagens bruscas. Estas comunicações são feitas através de mensagens, enviadas pelos nós vizinhos, sendo possível a cada equipamento inserido em cada veículo efetuar autonomamente decisões de navegação, proporcionando melhores opções de segurança.

Para além disso, a comunicação entre veículos permite uma panóplia de novas aplicações, como a disseminação de informação baseada na localização do veículo, redes sociais e distribuição de jogos interativos [1].

Assim, torna-se importante existir uma forte investigação nesta área, de modo a alcançar

uma conetividade *wireless* e transmissão de informações estável e segura. Para tal, é importante fornecer os meios para que esta investigação possa ser conduzida de forma mais eficiente e robusta. Neste sentido, para possibilitar a experimentação e teste das funcionalidades e das capacidades das redes veiculares, é feita uma aplicação desta rede veicular num ambiente real, à qual se dá a demoninação de *testbed*. Nesta *testbed* é necessário encontrar uma forma de conseguir que a execução de experiências seja efetuada de forma mais simples, obtendo-se os resultados destas experiências de forma automática, de modo a simplificar o trabalho dos investigadores. É ainda importante garantir uma gestão eficaz dos recursos desta *testbed*, assim como o seu controlo de forma remota e gestão dos utilizadores e das suas comunicações, como, por exemplo, o instante em que estes recursos se encontram em alcance de comunicação e em que posições geográficas, informações das comunicações realizadas e das suas características, com tempos de início e fim, etc., sendo estas as motivações deste trabalho.

1.2 Enquadramento

O projeto DRIVE-IN, em parceria com a Universidade de Carnegie Mellon e suportado pela Fundação para a Ciência e Tecnologia (FCT), tem por objetivo a pesquisa e o desenvolvimento das comunicações entre veículos. Este projeto envolve também a participação de investigadores de duas universidades portuguesas (Universidade do Porto e Universidade de Aveiro), o Instituto de Telecomunicações (IT) e ainda o Instituto da Mobilidade e dos Transportes Terrestres (IMTT), a NDrive e a RadiTáxis (Cooperativa dos Rádio-Táxi do Porto) [1] [2].

No projeto DRIVE-IN os conceitos, metodologias e as tecnologias deverão ser desenvolvidades em três vertentes principais, sendo estas: protocolos *Vehicular Ad-hoc NETwork* (VANET) geo-otimizados, navegação inteligente e colaborativa de carros e aplicações e serviços para VANET.

A indústria automóvel tem vindo a desenvolver alguns projetos na área das VANET, sendo estes mais focados no melhoramento dos padrões de segurança, por meio das redes veiculares. Embora seja importante a investigação e o desenvolvimento das VANET neste sentido, existem também outras possibilidades menos exploradas das redes veiculares, as quais não têm recebido a mesma atenção, nem da comunidade científica, nem da indústria automóvel. Desta forma, o projeto DRIVE-IN tem como objetivo colmatar esta lacuna, focando-se não só na segurança, mas também noutros aspetos, como a otimização do tráfego e as aplicações de *infotainment* (informação e entretenimento) [1].

As VANET, como novas tecnologias de redes *wireless*, têm vindo a ser testadas em ambientes de pequenas plataformas experimentais em rede, i.e., *testbeds* [3] [4]. No entanto, no projeto DRIVE-IN, um dos objetivos é ter uma rede experimental de larga escala, o que ainda não existe atualmente. Desta forma, é necessário um sistema unificado para a gestão dos seus recursos, das experiências realizadas e dos resultados obtidos das mesmas. Portanto, é importante encontrar uma forma de garantir que a gestão dos recursos da *testbed* possa ser feita de forma eficiente, simples e, quando possível, automática, assim como garantir os meios para a execução de experiências de forma eficaz e simples e a recolha automática dos seus resultados, garantindo os meios para a sua consula e análise. Também é importante encontrar forma de fornecer os meios para a gestão de utilizadores da *testbed* e das suas comunicações.

1.3 Objetivos

Tendo por base a *testbed* do DRIVE-IN, sugiram novos desafios relativamente à sua usabilidade, como a gestão de todos os seus recursos, bem como a execução de experiências e recolha de resultados. Neste sentido, foram estabelecidos, *a priori*, alguns objetivos a ser cumpridos, no decorrer do projeto aqui apresentado:

- Encontrar uma solução para a distribuição de *software* na *testbed* remotamente, visto a dificuldade que a mesma apresenta quando os nós estão distribuidos em veículos;
- Fornecer os meios necessários para a criação e gestão de experiências para a *testbed*;
- Garantir a simplificação da execução de experiências remotamente na *testbed*;
- Encontrar uma solução para a recolha automática e autónoma de dados de experiências realizadas na *testbed*, permitindo poupar tempo e recursos de comunicação aos investigadores que a usam;
- Fornecer os meios para melhorar e facilitar todo o processo de gestão da *testbed*;
- Verificar o bom funcionamento do sistema de gestão a ser implementado na *testbed*;
- Verificar a comunicação eficiente entre nós com recurso à rede *wireless* da norma IEEE 802.11p, usando o sistema de gestão implementado;
- Fornecer os meios para a gestão de comunicações e utilizadores da *testbed*;

- Obter resultados de *upload* de *software* remotamente para a *testbed* com recurso ao sistema de gestão, assim como resultados de gestão de experiências e obtenção dos dados das medições das mesmas.

Tendo em conta o descrito anteriormente, o objetivo geral deste trabalho passa por obter um meio de gerir remotamente toda a informação da *testbed*. Com este trabalho de Dissertação pretende-se atingir uma gestão eficaz e remota de todos estes recursos, bem como a facilitação da experimentação na *testbed* e recolha de resultados. Desta forma, esta Dissertação contribui para expor a forma como um sistema de gestão pode ser implementado numa *testbed* de redes veiculares como a do DRIVE-IN. As contribuições desta Dissertação consistem no desenvolvimento do sistema de gestão que permite dar resposta aos objectivos propostos e descritos anteriormente.

1.4 Organização da Dissertação

A Dissertação presente é organizada da seguinte forma:

- Capítulo 1: expõe a motivação, o enquadramento e os objetivos definidos para este trabalho de Dissertação;

- Capítulo 2: apresenta o estado de arte das redes veiculares, o tema em que este trabalho se insere, desde a sua definição até às principais aplicações, passando pelas características técnicas. De seguida, são discutidos os vários sistemas de gestão de *testbed* considerados para o cumprimento dos objetivos propostos;

- Capítulo 3: fornece uma explicação detalhada do sistema de gestão utilizado neste trabalho de Dissertação, o qual foi aplicado na rede veicular do DRIVE-IN;

- Capítulo 4: descreve a forma como o sistema de gestão foi implementado para a *testbed* do DRIVE-IN, desde a sua arquitetura, passando pela instalação e implementação das funcionalidades necessárias para a sua aplicação na *testbed*, bem como o trabalho realizado para a gestão das comunicações e de utilizadores da *testbed*;

- Capítulo 5: apresenta os resultados obtidos com o sistema de gestão, demonstrando o seu bom funcionamento;

- Capíutlo 6: expõe as conclusões alcançadas após a execução deste trabalho, assim como um resumo de todo o trabalho realizado, fornecendo também a descrição das tarefas que ainda devem ser efetuadas no âmbito de trabalho futuro.

Capítulo 2

Estado de Arte

Os veículos têm vindo a incorporar cada vez mais avanços tecnológicos, que melhoram a experiência de condutores e passageiros. Alguns exemplos são a utilização de sistemas de travagem, sensores de estacionamento e de proximidade de outros veículos ou alarmes de velocidade, que indicam quando é ultrapassado o limite permitido. Em geral, estes sistemas são baseados em sensores e atuadores cada vez mais sofisticados, que permitem ao veículo detetar sinais no ambiente e informar o condutor, sendo que estes sistemas são restritos à interação entre o veículo e o condutor [5].

O próximo passo da evolução tecnológica consiste nos sistemas de comunicação que permitam interação entre os diferentes veículos, o que tem atraído atenções consideráveis não só da comunidade científica, mas também da indústria automóvel, uma vez que estes desenvolvimentos contribuem para um Sistema Inteligente de Transportes (*Intelligent Transportation System* (ITS), demonstrado na figura 2.1) [6][5]. Este pode conduzir a melhorias importantes no desempenho operacional da rede de transportes, na segurança de todos os utilizadores da estrada, seja como condutores ou peões, no conforto da mobilidade, e pode mesmo contribuir para o aumento da produtividade e expansão do crescimento económico e de emprego e redução de impactos ambientais [5].

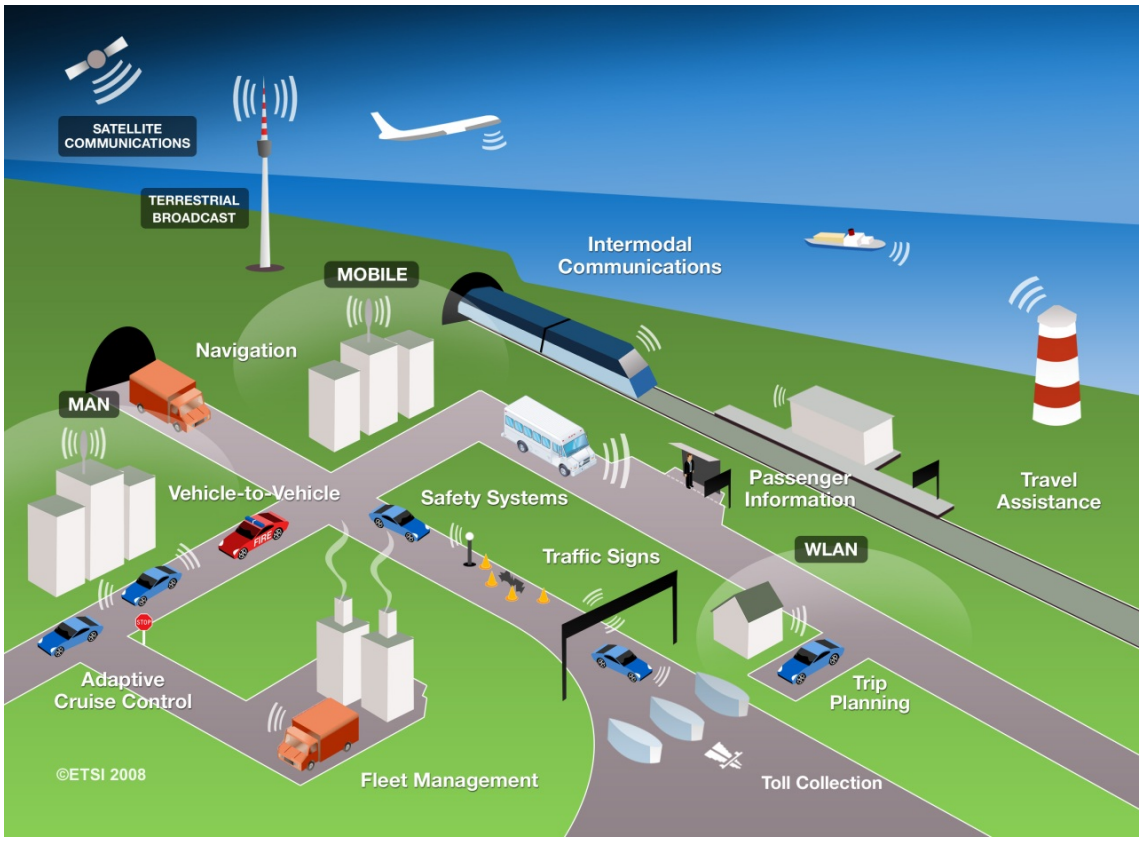


Figura 2.1: Gama de aplicações do ITS [7]

No decorrer do desenvolvimento deste trabalho, foi inicialmente necessário um enquadramento teórico, que teve por base principalmente o livro [8], seguindo-se uma análise e estudo de artigos relacionados com o tema, de modo a descobrir os desenvolvimentos já alcançados até ao momento. Posteriormente, procedeu-se à investigação dos sistemas de gestão para *testbed*, de forma a encontrar a melhor opção para a implementanção em redes veiculares.

Neste capítulo são inicialmente descritas as redes veiculares, as suas características e possíveis aplicações e benefícios que estas redes possam trazer à comunidade em geral, seguindo-se uma análise dos protocolos existentes. De seguida, é efetuada uma análise dos Sistemas de Gestão de *testbed*, para a sua aplicação na rede veicular do DRIVE-IN.

2.1 Redes *Wireless* e Tecnologias

Nos últimos anos, as tecnologias de comunicação *wireless* tem proporcionado enormes vantagens, pois permitem grande mobilidade, de forma mais económica. Uma rede *wireless* consiste então numa rede em que os nós têm algum tipo de mobilidade, sem eliminar a

possibilidade de existirem também nós fixos.

Atualmente, as tecnologias *wireless* podem ser divididas em dois grupos principais. Por um lado, existem tecnologias de grande área como GSM, GPRS ou UMTS, que têm uma largura de banda moderada. Por outro lado, existem as tecnologias de área local como *Wireless Local Area Network* (WLAN) com muito maior largura de banda. Existem duas normas diferentes para WLAN: HIPERLAN da *European Telecommunications Standards Institute* (ETSI) e 802.11 do *Institute of Electrical and Electronics Engineers* (IEEE). Atualmente, a norma 802.11 é a mais dominante [9].

A rede *wireless* pode ser classificada como infraestruturada ou em modo *ad hoc*. Relativamente ao primeiro caso, a comunicação entre dispositivos móveis ocorre através de nós fixos, como é o caso das redes celulares, sendo assim necessário recorrer a infraestrutura. No segundo caso, não é necessário recorrer a infraestrutura uma vez que a rede é composta por dispositivos móveis, que comunicam entre si. As redes móveis *ad hoc* têm o nome de *Mobile Ad hoc Network* (MANET) [9][10][11].

O IEEE criou as normas IEEE P1609.1, P1609.2, P1609.3 e P1609.4 [12] para as redes veiculares. A primeira, P1609.1, é a norma para a gestão de recursos do acesso *wireless* para redes veiculares, ou *Wireless Access in Vehicular Environments* (WAVE). Ela define os serviços e interfaces da aplicação de gestão de recursos do WAVE, bem como os formatos de mensagens de dados; fornece também acesso para as aplicações para as outras aquiteturas. O segundo, o P1609.2, define a segurança, formatação de mensagens segura, processamento e troca de mensagens. O P1609.3 define os serviços de encaminhamento e de transporte, assim como a gestão da informação base para a camada de protocolos. Por fim, o P1609.4, lida principalmente com a especificação de múltiplos canais na norma *Dedicated Short-Range Communications* (DSRC).

A *stack* protocolar WAVE usa uma versão modificada do protocolo IEEE 802.11a, conhecido como IEEE 802.11p, para o seu protocolo *Medium Access Control* (MAC). Como a norma IEEE 802.11 foi pensada para pouca mobilidade, o IEEE 802.11p endereça problemas importantes como as desconexões frequentes e *handoff*. Esta norma define modificações ao IEEE 802.11 para suportar aplicações para redes veiculares, que incluem troca de dados entre veículos em altas velocidades, e entre veículos e infraestruturas na estrada no espectro DSRC. Atualmente, a norma IEEE 802.11p tem por objetivo providenciar um conjunto mínimo de especificações requeridas para garantir a interoperabilidade entre dispositivos *wireless*, que tentam comunicar em ambientes de potenciais mudanças rápidas e em situações onde as tro-

cas de informações devem ser completadas num tempo muito mais curto, que naquele da infraestrutura ou das redes *ad hoc* do 802.11.

O *WAVE mode Basic Service Set* (WBSS) no IEEE 802.11p melhora as funções MAC do IEEE 802.11 para ambientes de comunicação de alterações rápidas. As estações móveis do WAVE tornam-se membros de um WBSS de uma de duas maneiras, podem atuar como um *provider* ou então como um *user*. O objetivo mais importante é que estas estações móveis se juntem à rede veicular e transmitam ou recebam dados tão rapidamente quanto possível. Devido a estes ambientes de mudanças rápida, o *provider* WBSS e o *user* devem estar prontos para que estas comunicações decorram tão rapidamente quanto possivel. Deste modo, o WBSS não requer autenticação e associação da subcamada MAC para que seja permitida a transmissão de dados; num WBSS, o *user* apenas necessita de receber o anúncio de um *provider* antes de iniciar as transmissões [13].

2.2 Redes Veiculares

2.2.1 Definição

As redes veiculares são uma classe de rede *wireless*, que sugiram com o avanço das tecnologias *wireless* e indústria automóvel, permitindo assim uma comunicação entre diferentes veículos, sem ser necessário recorrer a uma infraestrutura física. As redes veiculares formam-se espontaneamente entre veículos em movimento, que estão equipados com interfaces *wireless* e que podem corresponder a tecnologias homogéneas e heterogéneas. Estas redes são demoninadas VANET e são uma classe particular de MANET, na qual os nós são veículos que comunicam com outros veículos ou com equipamentos na estrada, os *Road Side Unit* (RSU) (figura 2.2). Assim, no caso das VANET, os nós são móveis, podendo atingir grandes velocidades e diferentes trajetórias, de acordo com os limites das vias públicas [14][6].

Como já mencionados anteriormente, os veículos podem ser privados, pertencentes a indivíduos ou empresas privadas, ou podem ser meios de transporte públicos, como autocarros, táxis, carros da polícia, etc.. O equipamento fixo pode pertencer ao governo, a operadores de rede privados ou a prestadores de serviços [6].

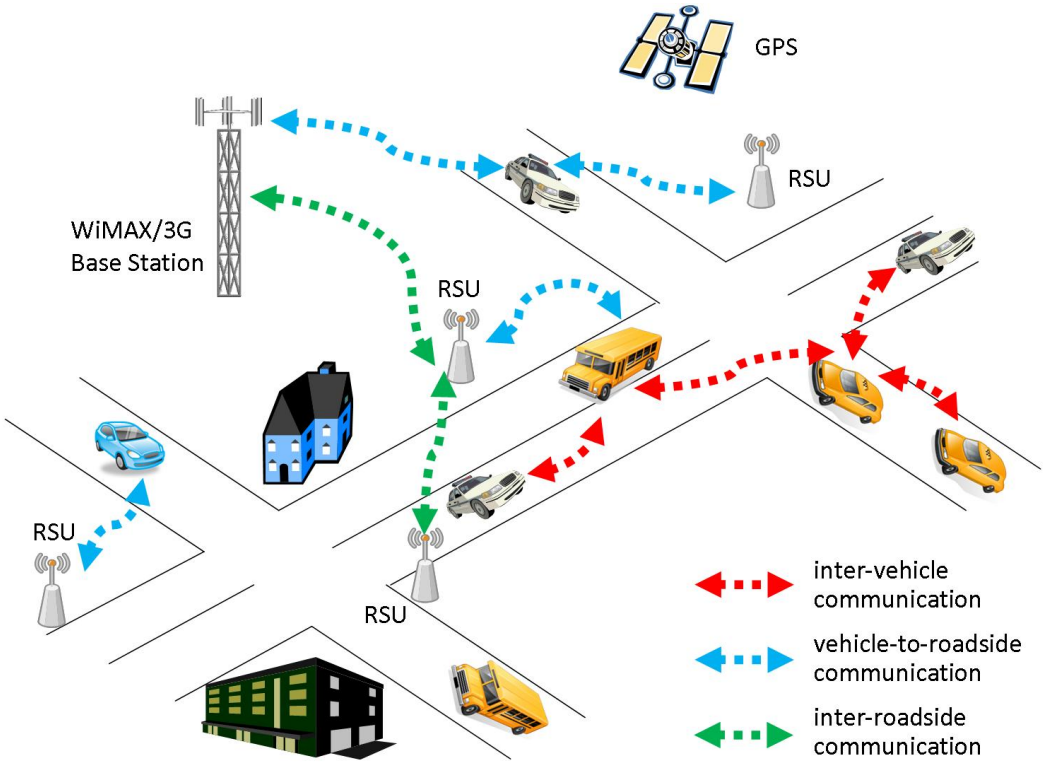


Figura 2.2: Diferentes tipos de comunicação em redes veiculares [15]

O grande interesse pelas redes veiculares é demonstrado não só pela comunidade científica e indústria automóvel, como já mencionado anteriormente, mas também pelas autoridades governamentais e pelas organizações responsáveis pela normalização. Neste contexto, surgiu na América do Norte o DSRC, onde, em 2003, foram aprovados 75 MHz de espectro pela U.S. *Federal Communication Commission* (FCC), reservados a este tipo de comunicações, que visa principalmente as redes veiculares. Por outro lado, foi iniciado na Europa o *Car-to-Car Communication Consortium* (C2C-CC) [16] pelos fabricantes de automóveis, com o objetivo de melhorar a segurança e eficiência do tráfego com recurso à comunicação entre veículos. Também o IEEE avançou com uma na família de normas 1609 para o acesso *wireless* em ambientes veiculares, como referido anteriormente [6].

As redes veiculares possuem ainda uma série de desafios à sua adoção em larga escala. O principal desafio está relacionado com a elevada mobilidade dos nós, o dinamismo dos cenários e a escabilidade dos seus mecanismos em termos do número de nós com que conseguem funcionar corretamente. A perda de conetividade durante a transmissão de dados e o tempo reduzido em que dois nós permanecem em contacto são outros desafios. E, neste cenário, os protocolos criados para as MANET não são adequados.

2.2.1.1 Arquiteturas das Redes Veiculares

Os recentes avanços em tecnologias de rede *wireless* permitem uma implantação de arquiteturas de redes veiculares em ambientes rurais, citadinos e rodoviários. Tais arquiteturas devem permitir a comunicação entre veículos nas proximidades e entre veículos e o equipamento fixo na estrada.

As três alternativas de arquitetura são (figura 2.3):

- Uma rede *ad hoc* pura, veículo para veívulo (*Vehicle-to-Vehicle* (V2V));
- Infraestruturada (*Vehicle-to-Infrastructure* (V2I));
- Híbrida (*Vehicle-to-Road* (V2R)) [6].

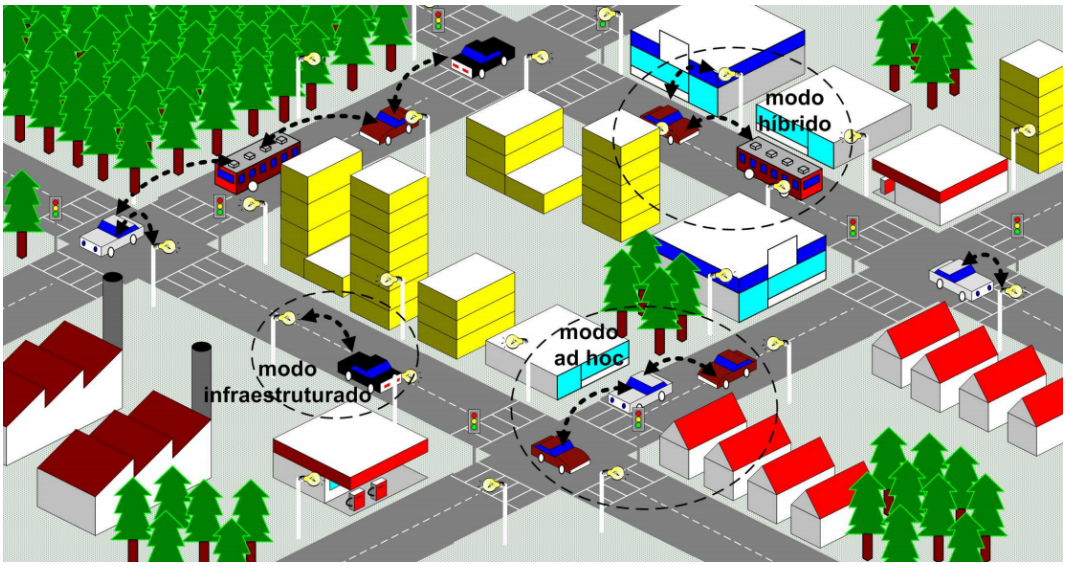


Figura 2.3: Diferentes arquiteturas possíveis num cenário de redes veiculares [5]

Na arquitetura *ad hoc*, os veículos comunicam entre si sem qualquer suporte externo ou elemento, enquanto os veículos se comportam como *routers*, e são responsáveis pela recolha e encaminhamento de informações críticas. Esta configuração é a mais simples, por não exigir nenhum tipo de infraestrutura, mas tem como desvantagem o facto da conectividade de rede depender da densidade e do padrão de mobilidade dos veículos.

Para evitar problemas de conetividade, a arquitetura infraestruturada emprega nós estáticos e distribuídos ao longo das ruas ou estradas (RSU). Estes nós estáticos funcionam como pontos de acesso da rede IEEE 802.11, também em modo infraestruturado, e servem como

intermediários de comunicação. A grande vantagem do modo infraestruturado é o aumento da conectividade e a possibilidade da comunicação com outras redes como, por exemplo, a Internet. No entanto, a conectividade da rede só pode ser garantida mediante um número elevado de elementos fixos, o que pode elevar, de forma considerável, o custo da rede.

A arquitetura híbrida corresponde a uma solução intermediária ente *ad hoc* e infraestruturada. Nesta arquitetura, é utilizada uma infraestrutura mínima para aumentar a conectividade da rede e fornecer serviços [5]. Neste último caso, os veículos podem comunicar com a infraestrutura tanto em um *single-hop*, como em *multihop*, de acordo com as posições dos veículos em relação ao ponto de ligação com a infraestrutura. Na verdade, a arquitetura V2R inclui implicitamente a comunicação V2V [6].

Dentro do C2C-CC é proposta uma arquitetura de referência para as redes veiculares, distinguindo-se então três domínios: do veículo em si, *ad hoc* e das infraestruturas, como mostra a figura seguinte.

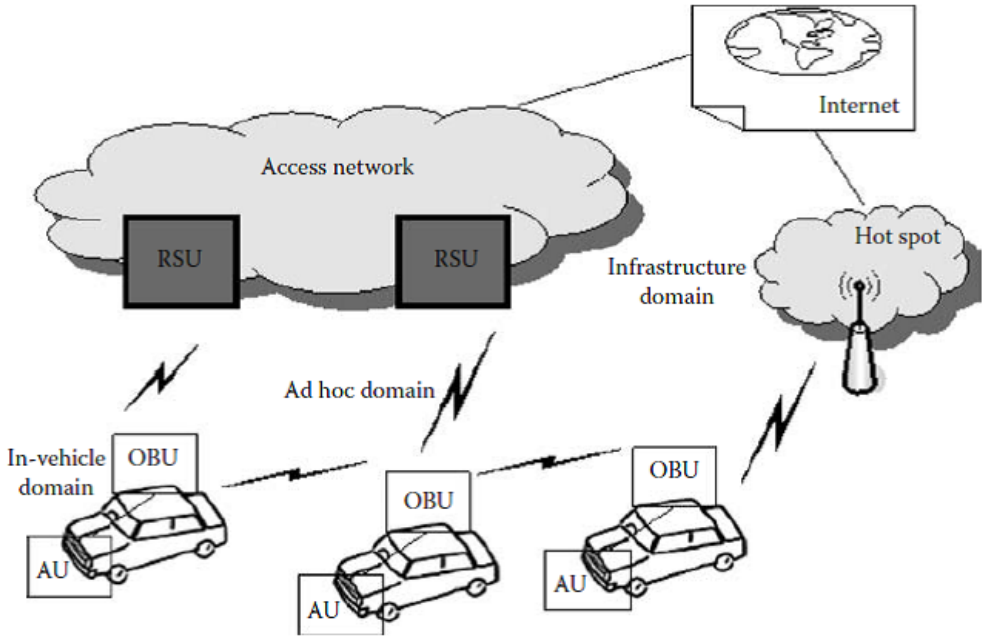


Figura 2.4: Arquitetura de referência do C2C-CC [6]

O domínio do veículo refere-se a uma rede local dentro de cada veículo composta por duas unidades: *On Board Unit* (OBU) (cujos componente são indicados mais à frente, na secção 2.2.2.1) e uma ou mais *Application Unit* (AU). Uma OBU é um dispositivo no veículo com capacidades de comunicação (com ou sem fios), enquanto que uma AU é um dispositivo que

executa uma única ou um conjunto de aplicações, ao fazer uso das capacidades de comunicação do OBU. Com efeito, uma AU pode ser integrada como parte de um veículo, e ser permanentemente ligada a uma OBU. Por outro lado, a AU também pode ser um dispositivo portátil, como um *Personal Digital Assistant* (PDA) que pode ser dinamicamente ligada e desligada de uma OBU. A AU e a OBU são normalmente ligadas por cabo, embora a ligação *wireless* também seja possível (usando, por exemplo, Bluetooth).

O domínio *ad hoc* é uma rede composta por veículos equipados com OBU e por RSU dispostas ao longo das estradas e, assim, as OBU, de diferentes veículos, formam uma rede VANET híbrida. As OBU e RSU podem ser vistas como nós de uma rede *ad hoc*, móveis e estáticos, respetivamente. Uma RSU pode ser ligada a uma rede de infraestrutura que, por sua vez, pode ser ligada à Internet. Estas podem comunicar diretamente ou por *multihop*, e a sua principal função é a melhoria da segurança rodoviária, executando aplicações especiais pelo envio, receção ou transmissão de dados no domínio *ad hoc*.

Quanto ao domínio das infraestruturas, existem dois tipos de acesso às mesmas: RSU e *hot spot*. As RSU permitem à OBU aceder à infraestrutura e, consequentemente, ser ligada à Internet. A OBU também pode aceder à Internet via Wi-Fi *hot spots* públicos, comerciais ou privados. Na ausência de RSU e de *hot spots*, a OBU pode utilizar recursos de comunicação de redes celulares (*Global System for Mobile communications* (GSM), *General Packet Radio Service* (GPRS), *Universal Mobile Telecommunications System* (UMTS), WiMAX e 4G) caso a OBU esteja integrada com os mesmos [6].

2.2.1.2 Características específicas das redes veiculares

As redes veiculares têm características e comportamentos inerentes que as distinguem facilmente de outro tipo de redes móveis. Em comparação com outro tipo de redes de comunicação podemos referir as seguintes vantagens [17]:

- Bateria ilimitada: as questões relacionadas com a bateria de dispositivos móveis, não são geralmente uma restrição significativa em redes veiculares, uma vez que o nó (veículo) em si, enquanto ligado, pode fornecer energia contínua para a computação e comunicação dos dispositivos.
- Maior capacidade computacional
- Mobilidade previsível: ao contrário das redes móveis *ad hoc* clássicas, onde é difícil prever a mobilidade dos nós, os veículos tendem a ter muitos movimentos mais previsíveis

que são (geralmente) limitados às estradas. A informação sobre estrada é muitas vezes disponível a partir de tecnologias de sistemas de posicionamento baseados na localização, como o *Global Positioning System* (GPS). Dada a velocidade média, a velocidade corrente e a trajetória do veículo, é possível prever a sua posição futura.

No entanto, as redes veiculares têm que lidar com algumas características desafiadoras, que incluem [18]:

- Potencialidade a larga escala: ao contrário da maioria das redes *ad hoc* estudadas, em que se assume uma rede de dimensão limitada, as redes veiculares devem-se estender à totalidade de uma rede rodoviária, que poderá incluir bem mais participantes do que o normalmente estudado.

- Mobilidade elevada: o ambiente em que as redes veiculares operam é extremamente dinâmico e inclui configurações extremas, tendo que se adaptar e funcionar corretamente em qualquer cenário. Por exemplo, em autoestrada poderão existir velocidades relativas acima dos 300 km/h, em que a densidade de veículos poderá ser entre 1 a 2 veículos por quilómetro. Em meio urbano, a velocidade relativa poderá ser maior que 60 km/h, com uma grande densidade de veículos, principalmente nas horas de ponta.

- Rede fragmentada: as redes veiculares poderão ser frequentemente fracionadas, devido à natureza dinâmica do tráfego, que, em cenários de baixa densidade populacional, pode causar lacunas (áreas sem comunicação). Ou seja, a topologia da rede muda frequentemente, podendo levar à desconexão entre os nós em cenários de densidade reduzida de veículos, o que leva à impossibilidade do estabelecimento de comunicação entre os nós.

- Conectividade e topologia da rede: os cenários das redes veiculares são diferentes em relação aos das redes *ad hoc* clássicas. Como os veículos então em movimento e mudam a sua posição constantemente os cenários são muito dinâmicos. Assim, a topologia muda frequentemente, à medida que a conectividade entre os nós são ligadas/desligadas. De facto, a conectividade dos nós irá depender de dois grandes fatores: o alcance das comunicações *wireless* e a fração dos veículos participantes, onde apenas uma fração de veículos na estrada estão equipados com interface *wireless*.

2.2.2 Conceitos Básicos

2.2.2.1 On Board Unit

Assume-se que os veículos que fazem parte das redes veiculares têm incorporado os seguintes equipamentos:

- Um *Central Processing Unit* (CPU), que implementa as aplicações e protocolos de comunicação;
- Um transmissor *wireless*, que transmite e recebe dados de e para os veículos vizinhos e estrada;
- Recetor GPS, que permite um posicionamento relativamente preciso e informações de sincronização no tempo;
- Sensores apropriados para medir os vários parâmetros que têm de ser medidos e, eventualmente, transmitidos;
- Uma interface de entrada/saída que permite a interação humana com o sistema [19].

2.2.2.2 Endereçamento

Para a grande parte das aplicações é necessário um método de endereçamento. A maioria das redes veiculares pode ser classificada como redes *ad hoc*, o que significa que os nós (veículos e postos na estrada) se organizam numa rede. Portanto, podem ser utilizados os mesmos esquemas de endereçamento utilizados em outras redes *ad hoc*.

O endereçamento pode ser fixo, o que significa que cada nó tem um endereço fixo, atribuído por algum mecanismo, no momento em que o nó se junta à rede. O nó usa esse endereço enquanto parte da rede. Por outro lado, existe o endereçamento geográfico, em que cada nó é caracterizado pela sua posição e, à medida que o nó se move, o endereço muda. Existem atributos adicionais que devem ser utilizados para definir qual o endereço, como a direção do movimento do veículo, identificação da estrada, o tipo e características físicas do veículo ou mesmo características do condutor (e.g., iniciante ou profissional) [19].

2.2.2.3 Disseminação de Dados

O objetivo das redes veiculares é a disseminação de dados entre veículos, podendo incluir as estações nas estradas.

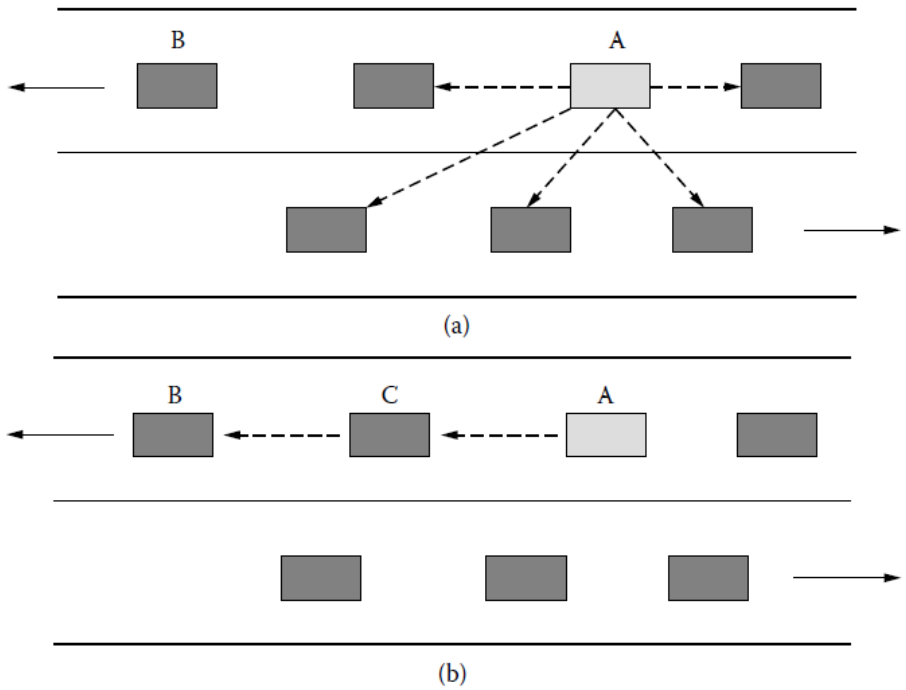


Figura 2.5: Transferência de dados single-hop (a) e multihop (b) [19]

A divulgação de dados pode ser *single-hop* ou *multihop* (figura 2.5). A transferência de dados *single-hop* é normalmente implementada com transmissão na camada MAC, onde o veículo A (na figura 2.5 em (a)) pode enviar uma mensagem só para os carros que estão na sua gama de transmissão (por exemplo, o veículo B nunca recebe a mensagem). A divulgação *single-hop* é também utilizada quando existem RSU na estrada que controlam a comunicação.

Já a disseminação de dados *multihop* está intimamente relacionada com as VANET. Os dados são transmitidos em vários saltos, onde veículos atuam como *relays* intermediários, como pode ser visto na figura 2.5 (b), onde o veículo C pode transmitir a mensagem para nós que não estão na faixa de transmissão (por exemplo, o veículo B pode também receber a mensagem). Portanto, este sistema requer uma camada de rede capaz de encaminhamento *multihop*.

Existem também variantes híbridas, onde, por exemplo, os dados são divulgados em *multihop* para a RSU mais próxima, que depois transmite os dados relevantes para veículos próximos, utilizando o sistema *single-hop* como método de divulgação [19].

Em segundo lugar, os dados podem ser divulgados em *unicast*, *multicast* ou *broadcast*. Em *unicast* existe apenas um emissor e um recetor dos dados. É necessário ter em conta que

o modo de transmissão é independente do esquema de endereçamento, e o endereçamento geográfico também pode ser utilizado em *unicast*. Em *multicast* há apenas um emissor de informação, mas podem existir vários recetores. O remetente geralmente não sabe exatamente quantos recetores existem, mas apenas que envia os dados para um grupo de destino específico. Em redes veiculares existem diversas aplicações relacionados com segurança pública que exigem que os dados sejam disseminados para, por exemplo, todos os veículos numa área específica, que entejam a conduzir numa direção específica, o que corresponderia a um grupo *multicast*. Em *broadcast* os dados devem ser divulgados a todos os veículos. No entanto, uma vez que as redes veiculares se podem estender por todos os continentes, o *broadcast* é normalmente utilizado exclusivamente numa determinada área, a *Zone of Relevance* (ZOR) [20].

2.2.2.4 Tecnologias de acesso à rede

Hoje em dia, existem várias normas de comunicação que podem ser utilizadas no acesso à rede, para as aplicações das redes veiculares, mas o IEEE 802.11 é a LAN padrão mais comummente assumida nas redes veiculares. No entanto, tem sido demonstrado, com modelos de propagação realistas, que a *Bit Error Rate* (BER) do 802.11 pode ser muito alta. O projeto de 802.11a, em que é explorado este problema, foi otimizado para redes locais sem ou com baixa mobilidade. No entanto, nas redes veiculares, a mobilidade pode ser muito elevada, daí que tenha sido desenvolvido um novo protocolo, o IEEE 802.11p, que está em fase de implementação e se destina à comunicação veicular [19][13].

As redes celulares cobrem grandes áreas e podem ser uma boa solução para as redes veiculares nas situações em que os veículos estão fora de grandes cidades ou estradas. No entanto, estes sistemas não foram projetados para um grande número de utilizadores, durante longos períodos, em situações de muito tráfego. Estas redes *single-hop* dependem fortemente da infraestrutura centralizada (*base stations*) para coordenar as transmissões dos nós móveis. Assim, o principal argumento para a utilização das redes celulares para sistemas de comunicação veiculares é que a infraestrutura já existe. Além disso, os sistemas 3G suportam comunicações de longa distância, oferecem garantia de qualidade de serviços e são projetados para mobilidade de alta velocidade.

Outra tecnologia de acesso à rede com potencial corresponde ao Bluetooth, destinado principalmente para comunicação de curto alcance e para novos protocolos que podem ser projetados especificamente para redes veiculares [19].

2.2.2.5 Arquiteturas de Comunicação

O objetivo da arquitetura de comunicação é proporcionar infraestruturas de comunicação para uma vasta gama de aplicações. Estas aplicações baseiam-se geralmente no modelo de camadas *Open Systems Interconnection* (OSI), onde cada camada fornece determinadas funções. Para as redes veiculares, o principal desafio é que diferentes aplicações necessitam de qualidade de serviços muito diferente entre si, exigindo diferentes tecnologias de acesso à rede, esquemas de endereçamento e protocolos.

A arquitetura de comunicação WAVE baseia-se no IEEE 802.11p e IEEE P1609. Esta norma suporta aplicações baseadas em IP e não IP. As aplicações não baseadas em IP, como é o caso das mensagens de advertência, são suportados pelo protocolo WAVE de mensagem curta *WAVE Short Message Protocol* (WSMP). Já o IEEE 802.11p tem suporte para aplicações prioritárias, o que significa que informações de alta prioridade, como dados relacionados com anti-colisão, serão transmitidas com o mínimo de latência.

Existe também outra arquitetura de comunicação em desenvolvimento pela *International Organization for Standardization* (ISO), a *Continuous Air Interface for Long and Medium distances* (CALM). Esta arquitetura irá incluir a norma WAVE, proporcionando uma comunicação perfeita num ambiente de rede heterogéneo, com diversas tecnologias de acesso de rede. Além disso, suportará endereçamento fixo e geográfico [19].

2.2.2.6 *Geocasting*

O *geocasting* corresponde a um encaminhamento *multicast* baseado na localização. Em *geocasting* todos ou alguns veículos numa determinada área devem receber informação, sendo que a decisão sobre que veículos devem receber dados não se baseia nos endereços, mas em outros dados, como a localização, direção, destino da viagem ou velocidade [10].

O *geocasting* baseado em *broadcast* seletivo pode representar uma boa solução para as aplicações de segurança pública, uma vez que alcança um baixo teor de latência na transmissão dos dados. Este *broadcast* seletivo usa *flooding* sobre a camada MAC. No entanto, uma vez que o esquema de *flooding* puro facilmente pode causar problemas de transmissão, como *broadcast storm*, numa rede densa, por exemplo, uma auto-estrada na hora do ponta, é utilizado um mecanismo de encaminhamento inteligente. Cada veículo faz uma decisão local de transmissão de uma mensagem ou não. O objetivo é minimizar o número de retransmissões desnecessárias dos dados. Os veículos próximos da fonte podem, por exemplo, cancelar o seu encaminhamento se se aperceberem de um outro veículo mais afastado a efetuar

a retransmissão de dados [19].

2.2.3 Desafios Técnicos

Como foi referido, existe uma série de desafios técnicos a ser resolvidos, de modo a ser possível proceder à implementação das redes veiculares. De uma forma geral, a escalabilidade e interoperabilidade são duas questões importantes que devem ser resolvidas; os protocolos e os mecanismos utilizados deverão ser escaláveis para vários veículos e interoperáveis com as diferentes tecnologias *wireless*. De seguida, são discutidos alguns destes desafios [6].

2.2.3.1 Comunicação Confiável e Protocolos MAC

À semelhança das redes *ad hoc*, as redes veiculares baseiam-se em comunicação *multihop*, o que tem potencial para permitir que sejam criadas infraestruturas virtuais entre os veículos em movimento, em que a comunicação deixa de poder ser feita apenas com a intermediação de infraestruturas e passa a ser feita veículo a veículo.

Na verdade, a comunicação *wireless multihop* representa um grande desafio em relação à confiabilidade da comunicação. Consequentemente, são essenciais protocolos MAC eficientes no local, enquanto se adapta ao ambiente altamente dinâmico de redes veiculares, e considerando a prioridade da mensagem de algumas aplicações (por exemplo, advertências de acidentes). Apesar da dinâmica da topologia e da sua elevada mobilidade, deve existir baixa latência na comunicação entre os veículos, a fim de garantir o seguinte:

- A confiabilidade do serviço para aplicações relacionadas com a segurança, tendo em consideração a sensibilidade do tempo de transferência de mensagens;
- A qualidade e continuidade de serviço para aplicações não relacionadas com a segurança.

Além disso, os protocolos MAC devem ter em consideração a comunicação heterogénea que é suscetível de ocorrer entre as diferentes tecnologias *wireless* (por exemplo, Wi-Fi e GSM), em redes veiculares [6].

2.2.3.2 Encaminhamento e Disseminação

As redes veiculares diferem das redes *ad hoc wireless* convencionais, não só por sofrerem mudanças rápidas nas ligações de *wireless*, mas também por terem de lidar com diferentes tipos de densidades de veículos que constituem as redes [21]. Por exemplo, as redes veiculares

em autoestradas ou áreas urbanas são mais propensas a formarem redes altamente densas durante horas de ponta, enquanto que nas áreas pouco povoadas ou estradas rurais e nos períodos noturnos é natural que ocorra frequentemente fragmentação da rede. Para além disso, como já foi explicado anteriormente, as redes veiculares apresentam uma panóplia de aplicações, que vão desde a segurança ao lazer. Assim, o encaminhamento e algoritmos de difusão devem ser eficientes e devem-se adaptar às características das aplicações das redes veiculares, permitindo prioridades de transmissão de acordo com o tipo de aplicação [19].

A maioria da investigação nas redes veiculares centrou-se na análise de algoritmos de encaminhamento capazes de lidar com o problema de *broadcast storm* numa topologia de rede com densidade elevada [22][23]. Até agora, a penetração das redes veiculares é baixa e, dessa forma, as redes necessitam da existência de infraestruturas de suporte para implantação em larga escala. No entanto, no futuro, é esperado que estas redes tenham uma maior penetração, com um menor número de infraestruturas de suporte e, assim, é importante nesta situação considerar o problema das redes desconectadas, que é um desafio central a investigar no desenvolvimento de protocolos de encaminhamento, no sentido de garantirem uma comunicação fiável e eficiente, e que possam suportar uma grande diversidade de topologias de rede.

Quanto à divulgação das suas mensagem, os algoritmos de difusão devem depender da densidade da rede, bem como do tipo de aplicação. Por exemplo, a divulgação de mensagens em aplicações de segurança deve ser principalmente do tipo *broadcast*, de maneira a assegurar a propagação de mensagens para o conjunto de veículos de destino sem causar uma *broadcast storm*. Em aplicações não relacionadas com a segurança, a transferência de mensagens deve ser efetuada em *unicast* ou *multicast*, que é mais adequado [6].

2.2.3.3 Segurança

A segurança e a privacidade são as maiores preocupações no desenvolvimento e na aceitação dos serviços, pelo que a segurança não deve ser comprometida pela facilidade que existe de uso dos protocolos para descoberta de serviços. Por conseguinte, é importante que existam soluções inovadoras para comunicação segura entre os participantes, bem com um acesso ao serviço autorizado e seguro. Para melhorar o acesso das redes veiculares, estas soluções devem aproveitar duas situações. Primeiro, o conceito de autenticação e comunicação das redes *ad hoc multihop*, que permite comunicação segura enquanto aumenta a cobertura das infraestruturas com o mínimo de custos de implementação. Depois, deve-se ter em conta a autenticação baseada na distribuição. Assim, as arquiteturas de segurança apropriadas

proporcionam a comunicação entre veículos e permitem diferentes serviços de acesso.

Deve ser desenvolvido um conjunto de mecanismos de segurança adequados para qualquer ambiente de rede veicular, proporcionando confiança, autenticação de controlo de acesso e serviço de acesso de seguro autorizado. Neste contexto, a otimização da autenticação é importante para ser estudada para comunicações com ou sem o recurso a infraestruturas, com o objetivo de facilitar o processo de reautenticação que pode ocorrer durante a mobilidade do veículo.

Por outro lado, o comportamento de cada nó é uma questão importante que pode ameaçar a segurança da comunicação, assim como a prestação de serviços em redes veiculares, sendo, por isso, um aspeto a considerar. Devido ao ambiente aberto e dinâmico das redes veiculares, a cooperação dos nós é um aspeto importante que deve ser satisfeito. Em determinadas situações, os nós podem ser "egoístas" ao não fazer encaminhamento das mensagens no sentido de poupar energia e largura de banda, ou simplesmente devido a preocupações com a segurança e com a prioridade. Por consequência, devem ser desenvolvidos mecanismos apropriados para detetar o "egoísmo" de determinados nós e fazê-los cooperar com os restantes nós da rede [6].

2.2.3.4 Configuração de IP e Gestão de Mobilidade

O potencial da arquitetura da comunicação veículo - infraestrutura é promissor em permitir o acesso dos veículos à Internet, bem como a prestação de serviços relacionados com a Internet a condutores e passageiros. Neste panorama, a configuração do IP e gestão de mobilidade correspondem a dois desafios técnicos que podem ameaçar a qualidade e a continuidade dos serviços.

Até ao momento, não existe um padrão para a configuração automática de IP em redes *ad hoc* e, portanto, o problema torna-se complexo para redes veiculares. Existem trabalhos em progresso por organismos de normalização, com o objetivo de resolver este problema. Além dos esforços do *Internet Engineering Task Force* (IETF) através do grupo de trabalho do Autoconf [24], para o desenvolvimento de soluções IPv6 para redes *ad hoc*, todos os comités internacionais definem arquiteturas para a comunicação veicular com uma *stack* nativa de IPv6, incluída nas *stacks* de protocolos, ou seja, IEEE 1609, ISO TC 204 (CALM) [25], C2C-CC, e o recém-formado ETSI TC ITS [26] [6].

Quanto à gestão da mobilidade, este é um problema crucial para aplicações não relacionadas com a segurança, onde a disseminação de mensagens não é baseada em *broadcast*. Na verdade, a ausência de um mecanismo de gestão de mobilidade ameaça a comercialização

deste serviço nas redes veiculares, perdendo-se assim o benefício da arquitetura veículo - infraestrutura, já que todos os serviços relacionados com a Internet não iriam garantir nem qualidade de serviço nem a sua continuidade [6].

2.2.4 Evolução e Progresso

2.2.4.1 Atores principais

De uma forma geral existem muitos projetos internacionais e nacionais levados a cabo por ações governamentais, pela indústria e por instituições académicas direcionados para as redes veiculares. Estes projetos incluem consórcios como *Vehicle Safety Consortium* (VSC) (Estados Unidos), *Collision Avoidance Metrics Partnership* (CAMP) (Estados Unidos), C2C-CC (Europa), *Advanced Safety Vehicle* (ASV) *Program* (Japão) [30] e os testes em larga escala e os esforços de normalização do *Vehicle Infrastructure Integration Consortium* (VIIC), nos Estados Unidos.

O programa dos Estados Unidos passa por efetuar testes operacionais e demonstrações, pesquisar tecnologias de apoio e aplicações para suporte à implementação, seguido do estudo da tecnologia de modo a perceber o seu potencial para as redes veiculares.

As aplicações alvo nos Estados Unidos incluem a segurança, eficiência do tráfego, pagamentos eletrónicos e gestão do relacionamento com o cliente. Por outro lado, na Europa esperam-se menos infraestruturas e assim as aplicações centrais, neste caso, serão a segurança e eficiência do tráfego. Neste contexto o C2C-CC (já referido anteriormente) foi iniciado por seis fabricantes europeus de automóveis (Audi [27], BMW [28], DaimlerChrysler [29], Fiat [30], Renault [31] e Volkswagen [32]), sendo esta uma organização sem fins lucrativos, com o objetivo de desenvolver um padrão de comunicação aberto entre veículos e para garantir a interoperabilidade entre os veículos da Europa, com recurso a uma tecnologia LAN *wireless* (WLAN IEEE 802.11).

Além disso, as empresas de telecomunicações que possuem grandes infraestruturas também direcionam uma atenção especial ao desenvolvimento das redes veiculares, tendo a vontade de participar no desenvolvimento da tecnologia através de parcerias com indústrias, universidades ou com as suas próprias equipas de investigação. Na verdade, eles vêem essas redes como uma evolução natural ou extensão dos sistemas *wireless* atuais, ao mesmo tempo que representam uma solução de baixo custo.

Muitas indústrias e empresas, envolvidas nos consórcios citados anteriormente, estão a investir de forma significativa para o desenvolvimento de novas soluções ITS. Alguns deles

(como, Dash, Google e TomTom) estão interessados em informação e entretenimento em tempo real e orientação dos condutores. Para além de mapear as estradas, permitem que os condutores ou passageiros recebam informações em tempo real sobre o tráfego, tenham oportunidade de se ligar à Internet e a obtenção de outras informações úteis (tais como postos de abastecimento de combustível, restaurantes e cinemas) durante o percurso.

Outro exemplo é a Microsoft, que propôs aos construtores e seus fornecedores uma nova versão do seu sistema operativo, o Windows Embedded Automotive 7, capaz de gerir todos os sistemas incorporados dentro do carro [33] [6].

Sendo uma tecnologia sempre em desenvolvimento, as redes veiculares pertencem às principais áreas de tópicos de pesquisa. Existe um grande número de conferências e *workshops* que lidam com esta área, e um conjunto de universidades e de institutos de investigação que trabalham no sentido de otimização dos diversos desafios encontrados à implementação das redes veiculares, entre os quais estão o encaminhamento e a disseminação de dados, segurança, entre outros [6].

2.2.4.2 Normalização

Em 1999, a U.S. FCC alocou 75 MHz do espectro de comunicação dedicado de curto alcance (DSRC) a 5,9 GHz (5,850-5,925 GHz) para ser utilizado exclusivamente pelas comunicações V2V e V2I na América do Norte, com o objetivo central de possibilitar aplicações de segurança, assim como melhorar o fluxo de trânsito.

No Japão, a banda de frequência alocada do DSRC foi dos 5,770 aos 5,850 GHz.

O canal 5885-5895 MHz é o canal de controlo que, geralmente, é restrito apenas a comunicações de segurança. Os dois canais que delimitam o espectro estão reservados às aplicações futuras para evitar acidentes e para aplicações públicas de segurança de alta potência. Os restantes canais são canais de serviço e estão disponíveis quer para aplicações de segurança, quer para os restantes tipos de aplicações.

Quanto à Europa, um obstáculo para introduzir VANET na segurança rodoviária foi a falta de um espectro de frequência específica. Em comparação com a América do Norte e Japão, o processo para a atribuição de frequências é consideravelmente complexo e demorado já que todos os países europeus e as respectivas autoridades nacionais estão envolvidos. A decisão pela Comissão Europeia para designar o espectro foi atribuída em Agosto de 2008. Está disponível a largura de banda dos 5875-5905 MHz para aplicações de segurança na estrada (com uma possível extensão de 20 MHz no futuro), e uma largura de banda dos 5855-

5875MHz para aplicações que não estão relacionadas com a segurança. A alocação de 50Mhz de frequência em conjunto com a possibilidade de expansão em 20 Mhz torna a alocação de espectro Europeia semelhante à alocação de 75Mhz existente na América do Norte [6].

2.3 Objetivos e Potenciais Aplicações das Redes Veiculares

As duas classes principais de aplicações das redes veiculares correspondem à segurança e otimização do tráfego e aplicações comerciais e de entretenimento, utilizando uma infinidade de tecnologias que colaboram entre si [14][19].

As aplicações direcionadas para a segurança em tempo real, para passageiros e condutores, consistem no objetivo inicial das redes veiculares [6]. Estas monitorizam a estrada circundante, aproximação de veículos, a superfície e curvas da estrada, etc.. Assim, os principais objetivos passam pela minimização de acidentes (salvando vidas humanas) e pelo melhorar a eficiência do trânsito (diminuindo o tempo de viagem), ao fornecer aos condutores avisos de colisão, alarmes sobre condições da estradas, vistas do trânsito, entre outras.

Já as aplicações comerciais e de conforto fornecerão ao condutor e/ou passageiro serviços ao nível do entretenimento, como acesso à Internet, pagamento eletrónico ou *streaming* de áudio e vídeo [14][19].

2.3.1 Segurança

Dentro das aplicações relativas à segurança, destacam-se a divulgação de informações sobre acidentes e condições da estrada (óleo, vias em mau estado, etc.) [14]. A principal característica que estas aplicações devem ter é a divulgação de dados de forma rápida e confiável, para um grande número de veículos, dentro de uma área de relevância. Portanto, o *geocasting* é o método comumente proposto de divulgação de dados para aplicações de segurança [19].

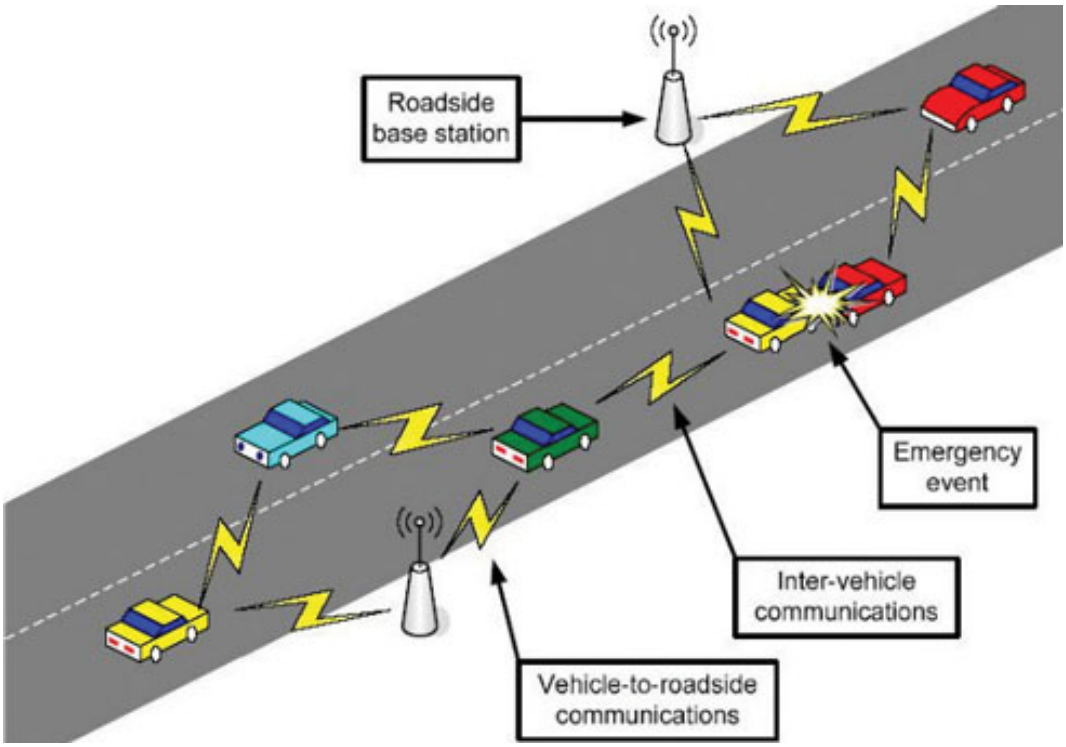


Figura 2.6: Exemplo de aplicação das redes veiculares [14]

A *Cooperative Collision Avoidance* (CCA) alerta pilotos potencialmente em rota de colisão, para que possam tomar decisões de condução. Assim, esta aplicação tem como objetivo evitar colisões em cadeia ou colisões frontais. Os veículos podem parar automaticamente ao receber uma mensagem de colisão ou sentir uma colisão, diminuindo a velocidade de veículos vizinhos. Esta abordagem também pode ser utilizada nos casos em que o condutor não possui visão completa da estrada. Assim, o motorista tem conhecimento de veículos que estão à sua frente em situações de neblina ou de chuvas intensas [14][6][34].

No caso das estradas maiores, nas horas de ponta, em caso de potencial colisão em cadeia, existem milhares de veículos que têm que ser informados, num curto período de tempo. Em estradas menores, o maior perigo são as colisões frontais, sendo que apenas um pequeno número de veículos está envolvido, mas com a agravante que normalmente as velocidades são maiores, uma vez que os veículos viajam em direções opostas. Assim, para esta aplicação deve ser utilizada uma rede *ad hoc multihop*, baseada na norma IEEE 802.11. As soluções baseadas na infraestrutura não se adequam, uma vez que não permitem tempos de latência suficientemente curtos. Assim, o método de disseminação proposto é algum tipo de difusão seletiva, em que apenas uma pequena área deve receber o aviso de modo a evitar a colisão

[34].

Esta aplicação exige informação em tempo real, confiabilidade de comunicação e tempos de latência extraordinariamente curtos (menor que 100ms [34]), o que corresponde a um desafio quando se considera a implementação desta aplicação.

No caso de uma aplicação de *Emergency Warning Message* (EWM), os veículos enviam avisos sobre acidentes ou condições da estrada perigosas (como curva perigosa ou descida súbita) para outros nós que se aproximem do local. Neste caso, pode ser necessário que a mensagem de aviso permaneça na área em questão, por um longo período de tempo [14][6]. Este tipo de aplicação, tem a vantagem de poder ser facilmente implementado com as tecnologias de acesso à rede atuais, seja baseada na infraestrutura, como as redes celulares, seja baseada numa rede *ad hoc*, como o IEEE 802.11. Existem duas categorias de EWM: EWM instantânea e EWM permanente [19].

No caso da EWM permanente, a mensagem de aviso deve ser divulgada a todos os veículos na zona de relevância (ZOR), que, neste caso, corresponde à área circundante. Um exemplo é uma aplicação que envia uma mensagem de aviso quando deteta que um veículo se encontra envolvido em um acidente, geralmente diminuindo rapidamente a velocidade.

Quando a mensagem é divulgada a todos os veículos na ZOR, que pode ser de vários quilómetros de diâmetro, a mensagem irá "desaparecer". Assim, tanto o IEEE 802.11 numa rede *ad hoc* ou uma rede celular podem ser utilizados como tecnologias de acesso à rede. No caso de uma rede *ad hoc*, é proposto um algoritmo de difusão seletiva para a divulgação de dados [35]. No caso de uma rede celular, o veículo envia uma mensagem de aviso para a sua *base station*, e, em seguida, a rede transmite a mensagem a todos os veículos na ZOR. No entanto, esta situação só é viável em situações de pouca densidade de veículos, pois este tipo de rede não está preparado para situações de grande fluxo de tráfego [19].

Em EWM com *geocasting* permanente, no início a mensagem é entregue a todos os veículos que se encontram na ZOR, como mostra a figura 2.7, usando, para tal, um algoritmo de *geocast*, sendo que os veículos com a informação devem detetar novos nós na ZOR, de modo a divulgar os respetivos dados [19][36].

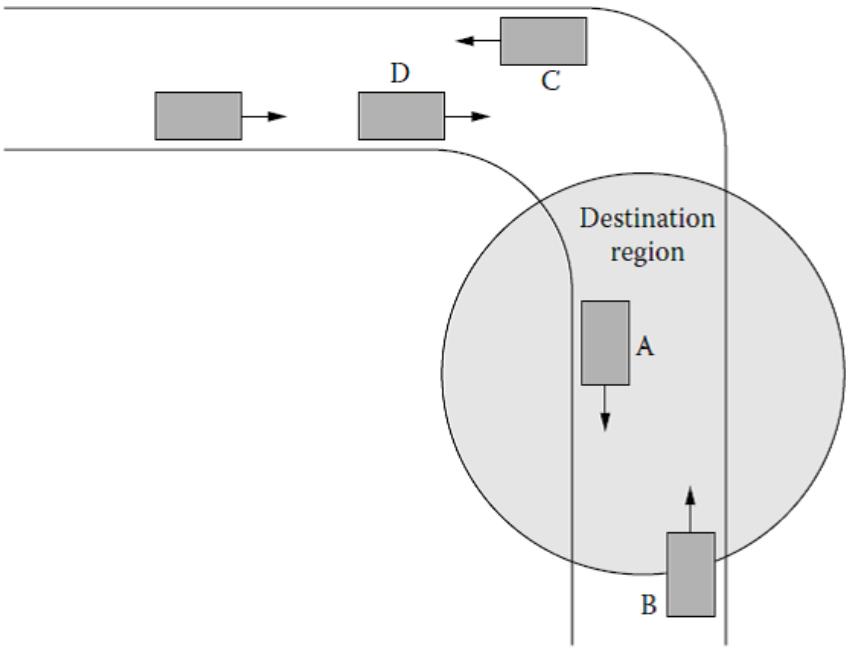


Figura 2.7: *Geocasting* permanente [19]

Um desafio para as aplicações EWM permanentes é garantir uma disseminação confiável de mensagens de alerta nas estradas com pouco tráfego, por exemplo, durante os períodos noturnos. Em [37], propõe-se que uma solução em que a mensagem é inicialmente transmitida a um servidor central. O servidor é, então, responsável pela divulgação da mensagem na região. Esta solução provavelmente requer uma infraestrutura celular, onde o veículo transmite o aviso de mensagem para a sua *base station*, que obviamente está dentro ou próximo da região. A *base station* pode facilmente transmitir a mensagem para todos os outros veículos que entram na região, porque estes estabelecem uma ligação com a respetiva *base station*.

Outra aplicação considerada é o *Slow/Stop Vehicle Advisor* (SVA) em que um veículo lento ou parado vai transmitir mensagem de aviso para os nós vizinhos, dando a conhecer a sua posição.

Noutra, como *Post Crash Notification* (PCN), um veículo envolvido num acidente transmite mensagens de advertência sobre a sua posição para veículos próximos, para que os mesmos possam tomar decisões de segurança atempadamente, bem como para as patrulhas rodoviárias, para fornecerem apoio [14][10][38].

A aplicação *Road Hazard Control Notification* (RHCN) está relacionada com a notificação entre veículos sobre situações de risco nas estradas, como deslizamentos de terra, buracos ou

gelo [38].

2.3.2 Otimização do Tráfego

A otimização do tráfego rodoviário tem como objetivo principal diminuir congestionamentos e os tempos de viagem. A principal diferença entre esta aplicação e as de segurança, descritas anteriormente, é a exigência de informação em tempo real destas últimas. Ou seja, a gestão de tráfego não tem tanta restrição ao nível dos atrasos. O propósito desta aplicação é informar o estado do trânsito na área circundante ou numa zona restrita (como um cruzamento) e, sendo assim, é tolerável que a informação demore um pouco mais de tempo a chegar [19].

A *Congested Road Notification* (CRN) deteta e notifica sobre congestionamentos rodoviários, informações estas que podem ser utilizadas para planeamento de rotas de viagem, para determinado destino [14], uma vez que, se o condutor receber uma mensagem a informar que a estrada que segue está congestionada, muito provavelmente ele tentará encontrar outro caminho [39].

Outro exemplo é um sistema de semáforos adaptados com base na comunicação *wireless* entre veículos e nós controladores fixos, nas áreas de intersecção (figura 2.8), o que melhoraria a fluência do trânsito nos cruzamentos e a diminuição de acidentes nestas regiões. Desta forma, os semáforos teriam uma imagem em tempo real, tomando decisões de acordo com os nós e podendo detetar situações de perigo [39]. Um cenário possível é o de veículos de emergência, por exemplo, ambulâncias, que podem comunicar com os semáforos, ficando assim luz verde em todos os cruzamentos [19].

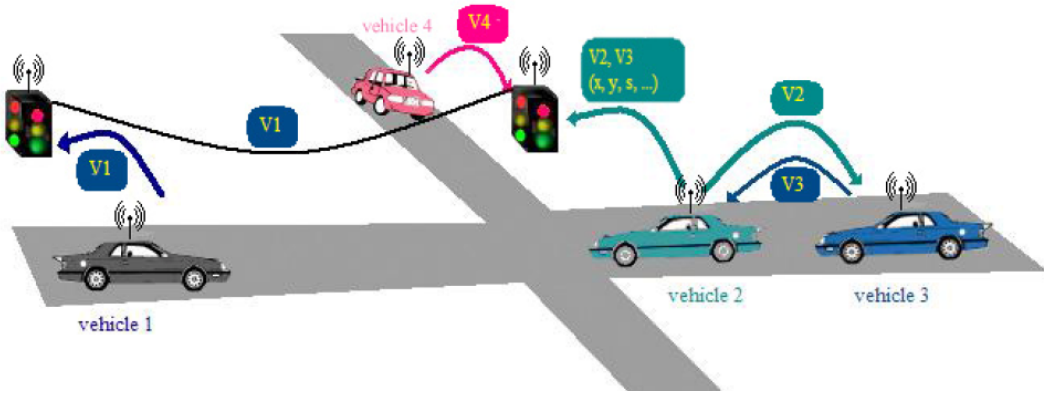


Figura 2.8: Semáforos e comunicação entre veículos para melhorar a eficiência do tráfego [39]

A aplicação para assistência nas intersecções do projeto DRIVE-IN pretende a migração dos semáforos atualmente existentes, como infraestruturas ao longo da estrada, para semáforos virtuais, que são transportados e implementados nos próprios veículos, utilizando no seu funcionamento as comunicações *Car-to-Car* (C2C), sem que seja necessário qualquer tipo de infraestrutura ao longo da estrada. Este tipo de sistema pode então melhorar significativamente o fluxo do trânsito nas interseções da estrada [40], comparativamente com os semáforos tradicionais, que aumentam o atraso no trânsito, em vez de melhorarem o fluxo, pois, a maioria dos semáforos utiliza tempos de ciclo fixos e parametrizados [41].

Com a comunicação C2C, quando os veículos de aproximam de uma intersecção, é possível verificar se há conflitos nos cruzamentos e, no caso de existirem, criam-se semáforos virtuais. Assim, um veículo eleito funciona como infraestrutura temporária do semáforo (sendo que este veículo terá luz vermelha, de modo a liderar os outros nós e deverá ser o veículo mais próximo do centro da interseção) e difunde, de seguida, mensagens para aos condutores que se aproximam do cruzamento, com a cor que lhes é fornecida, tendo os condutores acesso a essa informação dentro do próprio veículo.

Para este sistema funcionar, todos os veículos têm que estar equipados com os dispositivos essenciais para a implementação desta aplicação, o que leva a que a implementação seja gradual. Contudo, estudos efetuados na cidade do Porto demonstram que, com este tipo de tecnologia, o fluxo de trânsito poderia ser melhorado até 60 % [40].

A aplicação relacionada com a monitorização do tráfego fornece informações de trânsito localizadas, em tempo útil, numa grande extensão ao redor da posição atual do veículo. No entanto, assume-se que todos os veículos estão equipados com um sistema de navegação e com sensores que permitam a recolha de alguns dados, como a velocidade ou temperatura. Cada veículo recolhe dados, que são depois transmitidos, dentro de intervalos de tempo regulares, para uma ZOR adequada, que pode ter vários quilómetros de diâmetro. Os veículos que recebem os dados armazenam-nos depois numa tabela de valores dos vários segmentos da estrada. A informação armazenada pode então ser utilizada para simplesmente informar o condutor ou para melhorar o desempenho do sistema de navegação do veículo.

A monitorização do tráfego pode ser implementada com disseminação *multihop* ou com RSU que retransmitem dados provenientes dos veículos. Com um sistema *multihop*, o protocolo *geocasting* torna-se apropriado, podendo transmitir informações para todos os veículos na ZOR.

No entanto, é referido em [19] outro algoritmo de disseminação, que não é *multihop*. Em

intervalos regulares, cada veículo envia a sua tabela, com os dados agregados de cada segmento de estrada, para os veículos vizinhos (*single-hop*). Em [42] é feita uma comparação entre o algoritmo proposto e o *geocast*, que refere que o algoritmo proposto obtém melhores resultados para redes mais pequenas [42].

Existem ainda aplicações onde a troca de informações entre os veículos permite a execução segura de manobras no trânsito, como no caso de mudanças de faixa, em que as mensagens trocadas podem evitar colisões laterais [43].

Outra aplicação pensada está relacionada com a notificação sobre a disponibilidade de estacionamento, que demonstra a existência de lugares em estacionamentos numa determinada área geográfica [14][9].

2.3.3 Aplicações Comerciais e de Conforto

Para estas aplicações é utilizada a rede veicular para fornecer acesso à Internet, para uso dentro do veículo. Assim, existe uma variedade de redes de comunicação, tal como 2/3G, WLAN IEEE 802.11a/b/g/p e WiMAX, que podem ser exploradas para permitir novos serviços projetados para passageiros, aparte das aplicações de segurança, como aplicações de entretenimento.

Ficheiros multimédia podem ser transferidos para o sistema de entretenimento do veículo, podendo depois serem transmitidos ou visualizados no veículo. Assim o passageiro tem acesso a DVDs, músicas, livros aúdio, noticias, etc., a que pode recorrer durante a viagem. Para compartilhar músicas e filmes, utiliza-se um sistema C2C, em que os veículos trocam arquivos entre si [19].

O anúncio de serviços seria de particular interesse para os negócios na estrada, como bombas de gasolina, restaurantes de estrada, hotéis, entre outros, que poderiam assim anunciar os seus serviços aos motoristas na região de comunicação [14][9][10].

Outra aplicação possível das VANET são os jogos multijogador, que podem estar direcionados à Internet ou a equipamentos móveis, como telemóveis. Esta aplicação permite que os jogos tenham características relacionadas com a localização, proporcionando uma melhor experiência aos utilizadores [44].

2.3.4 Ultrapassagens Inteligentes

A utilização de VANET nas ultrapassagens permite que estas sejam feitas de forma informada, diminuindo o risco de acidente. Para esta aplicação, é utilizado um sistema que

se baseia não só nas redes veiculares, mas também numa tecnologia de *streaming* de vídeo: *See-Through System* (STS).

O sistema de transmissão de vídeo em tempo real melhora a visibilidade do motorista e apoia-o na decisão de ultrapassagem em situações desafiantes, tais como nas ultrapassagens de veículos longos/pesados, como camiões, em que existe pouca visibilidade da estrada. Existem já muitos veículos equipados com câmaras nos pára-brisas, que, por exemplo, leem sinais de trânsito e mostram depois essa informação digitalmente ao condutor. Com as redes veiculares, este equipamento pode transmitir para o condutor que queira realizar ultrapassagem, a imagem visual do troço da estrada à frente que não é visível. A utilização do STS proporciona assim ao condutor uma ferramenta adicional para determinar se condições do trânsito permitem iniciar uma manobra de ultrapassagem de forma segura [45].

Figura 2.9: Ultrapassagem inteligente [45]

Em relação às aplicações referidas, as redes veiculares podem abrir novas oportunidades de negócios para os fabricantes de automóveis, redes de operadores, prestadores de serviços e operadores em relação à implantação da infraestrutura.

Para aplicações relacionadas com a segurança, o operador de rede pode garantir a autenticação de cada participante, ao comportar-se como uma entidade de confiança, que autentica

os nós participantes, ou mesmo ter o papel de uma autoridade de certificação que emite um certificado para cada participante, de modo a provar a autenticidade dos nós durante a comunicação. Por outro lado, em aplicações não relacionadas com a segurança, operadores de rede e/ou prestadores de serviços fornecem acesso à Internet e serviços, podendo ter ainda o papel de autorizar o acesso a serviços e faturação dos serviços consumidos.

No entanto, existe ainda uma série de desafios técnicos a serem ultrapassados de modo permitir a evolução e implementação das redes veiculares em grande escala, alguns deles já desenvolvidos anteriormente [6].

2.4 Sistemas de Gestão de *Testbed*

À medida que as tecnologias de rede aparecem e evoluem, estas têm que ser avaliadas antes de serem incluídas nos produtos finais. Existem várias formas de o fazer, através de simuladores, como o NS3 [46] ou o OMNet++ [47], usando emulação, como no caso do Emulab [48], ou usando implementações de protótipo em ambiente reais, como está a ser realizado no caso específico do DRIVE-IN.

Tanto no caso das simulações como da emulação são utilizados modelos computacionais simplificados, o que faz com que a complexidade de ambientes reais não esteja presente na realização de experiências, refletindo-se nos resultados obtidos nas mesmas. Portanto, as experiências realizadas em ambientes reais são as que produzem resultados mais próximos da realidade. Por este mesmo motivo são utilizadas *testbeds*, que são consideradas uma alternativa eficaz, onde as novas tecnologias são testadas e controladas, num ambiente semelhante a um cenário real [1][49][50].

Com o intuito de cumprir os objetivos definidos para esta Dissertação, a tarefa inicial foi escolher um sistema de gestão de *testbed*. Os objetivos da utilização de um sistema deste tipo para a *testbed* do DRIVE-IN passam por permitir a distribuição de dados de forma automática, possibilitando ainda uma maior espontaneidade na execução de experiências na *testbed* e recolha dos dados resultantes das mesmas, o que possibilita uma maior prontidão na implementação das funcionalidades pretendidas. Outro requisito a cumprir com o sistema de gestão escolhido corresponde à gestão e controlo dos recursos da *testbed*, garantindo as faculdades necessárias para tal. Também seria necessário oferecer uma forma de gerir os acessos dos utilizadores às redes *wireless* disponibilizadas nos veículos. Neste sentido, o sistema de gestão escolhido terá que disponibilizar um método simples e escalável de definir as experiências e recolher os dados resultantes das mesmas, bem como facultar uma forma de

gerir os recursos da *testbed*, permitindo aos utilizadores da mesma obterem informação sobre estes recursos e possibilitar o seu controlo remotamente. Esta plataforma também deverá disponibilizar uma forma de se obterem as informações dos eventos ocorridos na *testbed*, especificamente com os acessos de utilizadores às redes *wireless* presentes nos veículos. Por outro lado, como não existe nenhum sistema de gestão de *testbed* apropriado para a norma IEEE 802.11p, utilizada na *testbed* do DRIVE-IN, o sistema de gestão escolhido deve ser compatível com as normas IEEE 802.11, tendo em conta a possível necessidade de realizar adaptações para assegurar o suporte à norma IEEE 802.11p. Assim, o respetivo sistema base deverá estar disponível livremente em código aberto, para ser possível proceder às modificações necessárias para adaptação à *testbed* do DRIVE-IN.

Existem vários sistemas de gestão de *testbed*, embora nem todos se adequem a este projeto. A realização de experiências em ambientes de *testbed* é mais complicada para os investigadores do que usando simulação, por exemplo. Existem muitas dificuldades na realização manual de experiências numa *testbed*, como o acesso aos nós remotamente, a sincronização de eventos e experiências, elementos externos que têm que ser considerados durante o planeamento da experiência, a execução das mesmas e análise dos resultados são algumas destas dificuldades. Adicionalmente, é necessária uma forma não ambígua para descrever e instrumentar uma experiência para que outros a possam repetir. Existe, portanto, necessidade de usar ferramentas mais eficientes e metodologias que forneçam descrições completas de experiências e as medições obtidas das mesmas [50]. De seguida, serão expostos alguns exemplos de sistemas de gestão de *testbeds*.

2.4.1 JAMES

O *JAva test-bed ManagEment System* (JAMES) [51] é um sistema completo de gestão de *testbeds* de redes de sensores *wireless*, que analisa e partilha resultados. Este sistema de gestão fornece uma pequena extensão de *software*, que deixa os protocolos a correr nos nós de sensores, para produzir *logs* de forma transparente, sem afetar o desempenho medido dos protocolos. Proporciona também um método padrão para descrever o formato de logs com recurso a ficheiros *Extensible Markup Language* (XML), nominados *Descriptors*. Por fim, fornece algumas ferramentas padrão para analisar resultados baseados nos XML *Descriptors* e apresentá-los de forma gráfica [51].

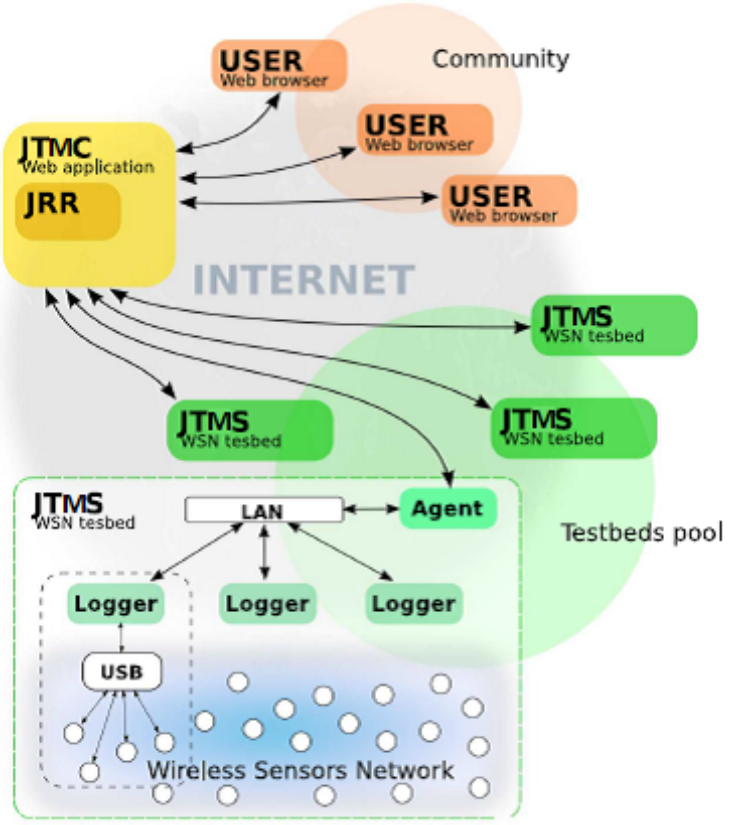


Figura 2.10: Arquitetura do JAMES [51]

Conforme apresentado na figura 2.10, o JAMES pode ser dividido em três subsistemas principais: o *JAMES Testbed Management System* (JTMS), o *JAMES Testbed Management Controller* (JTMC) e o *JAMES Results Renderer* (JRR). O JTMS é uma plataforma de *software* que permite o controlo de um ou mais nós, programação e configuração de nós de sensores, agendamento e execução de testes, recolha de *logs* e recuperação em caso de falha. O JTMS oferece aos utilizadores da *testbed* um canal de *debug*, que explora a interface *Universal Serial Bus* (USB) que vários nós de *Wireless Sensor Network* (WSN) usam para comunicar com sistemas externos. Para evitar as limitações da norma USB, o JTMS adota uma arquitetura escalável e hierárquica, que permite a gestão de *testbeds* de grandes dimensões, independentemente do número de nós que a WSN contém. O subsistema JTMS é implementado em dois tipos de componentes Java [52], nominados *Agents* e *Loggers*, respetivamente. Um *Logger* é um servidor Java que corre numa máquina ligada via uma cadeia USB até um subconjunto de nós da *testbed*. A sua função é gerir a sub-rede, instalando *software* nos nós, reiniciando, arrancando e parando nós ligados e obter os seus dados. Um conjunto inteiro de

nós pode ser gerido por mais de um *Logger*. Os PC que recolhem dados de um *cluster* de nós, via *Logger*, estão ligados via TCP/IP a um nó numa camada hierárquica mais alta que contém o componente *Agent*. Este último é outro servidor Java dedicado à *testbed*, que permite a diferentes utilizadores correr múltiplos testes na mesma *testbed*, controlar os *Loggers* e é, também, responsável pelo armazenamento e organização dos dados provenientes destes testes.

O JTMC é o núcleo do JAMES, este permite uma partilha remota de diferentes tesbeds baseadas em JTMS. O JTMC trata a interação com os utilizadores, fornecendo algumas funções padrão, como o arranque de testes sobre múltiplas *testbeds*, recolhendo e analizando resultados. Este componente é concretizado como uma aplicação *web* pública que permite a gestão de múltiplas *testbeds* através de JTMS. A qualquer altura, uma nova *testbed* pode ser adicionada, de forma a enriquecer a gama de *testbeds* disponíveis. O objetivo é aumentar as capacidades dos implementadores para analisarem o comportamento do seu protocolo em diferentes cenários.

Por fim, o JRR é um grupo de aplicações que tornam o JTMC capaz de construir e analisar os resultados gerados durante um teste. Ele permite gerar automaticamente gráficos que mostram o desempenho de um dado protocolo ou comparam o desempenho de diferentes protocolos. O JRR é concretizado por meio de um conjunto de classes Java, que o utilizador deve fornecer, de forma a produzir um *Renderer Plugin*. Cada um destes *plugins* é um *software* específico que é capaz de analisar *logs* em bruto e produzir alguma análise, como *throughput* ou latências. Os implementadores podem criar e partilhar com a comunidade de pesquisa qualquer *Renderer Plugin*, o que permite aos investigadores fazerem uma comparação com os seus protocolos [51].

2.4.2 MoteLab

O MoteLab [53] é um conjunto de ferramentas de *software* para a gestão de *testebeds* de redes de sensores cujos nós são ligados por *ethernet*, auxiliando-se de uma interface *web*, que permite aos utilizadores criar e agendar experiências. Este sistema automatiza a reprogramação da *testbed* e regista os dados gerados pelas experiências numa base de dados persistente. Os utilizadores podem recuperar os dados através da interface *web* ou diretamente da base de dados. O MoteLab também proporciona aos utilizados a opção de interagir diretamente com os nós individuais, no decorrer das experiências. Para além disto, o Motelab está disponível livremente em código aberto, daí que várias universidades e laboratórios de

investigação o utilizem como sistema de gestão das suas próprias *testbeds*.

O MoteLab contém um servidor central trata do agendamento, reprogramação de nós, *logging* de dados e provimento de uma interface para os utilizadores. Os utilizadores acedem à *testbed* usando um *browser web* para configurar ou agendar tarefas e descarregar dados.

O Motelab é constituído por vários componentes de *software*, sendo os principais:

- Base de dados MySQL [54]: armazena dados recolhidos durante experiências, a informação utilizada para gerar conteúdo *web* e o estado das operações na *testbed*. Quando uma conta de utilizador é criada, é também criada uma base de dados MySQL para o respetivo utilizador que guarda todos os dados das tarefas criadas pelo mesmo. Uma base de dados separada contém toda a informação relativa ao sistema de gestão, como informação de utilizadores ou estado dos nós da *testbed*;

- Interface *web*: uma interface em PHP [55] para criação de tarefas, agendamento e recolha de dados, bem como uma interface de administração para controlo das funcionalidades da *testbed*. Um utilizador comum, após fazer *login*, tem acesso a funcionalidades como um sumário de toda a informação disponibilizada. Outras páginas disponibilizadas estão relacionadas com a informação de utilizador, criação e edição de tarefas e ainda o seu agendamento;

- DBLogger: um *logger* de dados em Java para recolher e fazer o *parsing* dos dados gerados pelas tarefas que estão correr. Este *logger* é arrancado no início de cada tarefa, e liga-se a cada nó, usando TCP, faz o *parsing* das mensagens enviadas pela porta série e insere-as na base de dados MySQL apropriada;

- Job Daemon: um *script* em Perl [56] que corre como um *cron job* para configurar e desmantelar tarefas. Este *script* envolve reprogramação de nós e o arranque de outros componentes do sistema necessários, incluindo DBLogger e *serial forwarders*, terminando as tarefas quando acabadas, o que envolve parar a atividade do nó, matando os processos que eram necessários à tarefa. Por fim, efetua também o *dumping* dos dados da base de dados MySQL no formato apropriado para *download*.

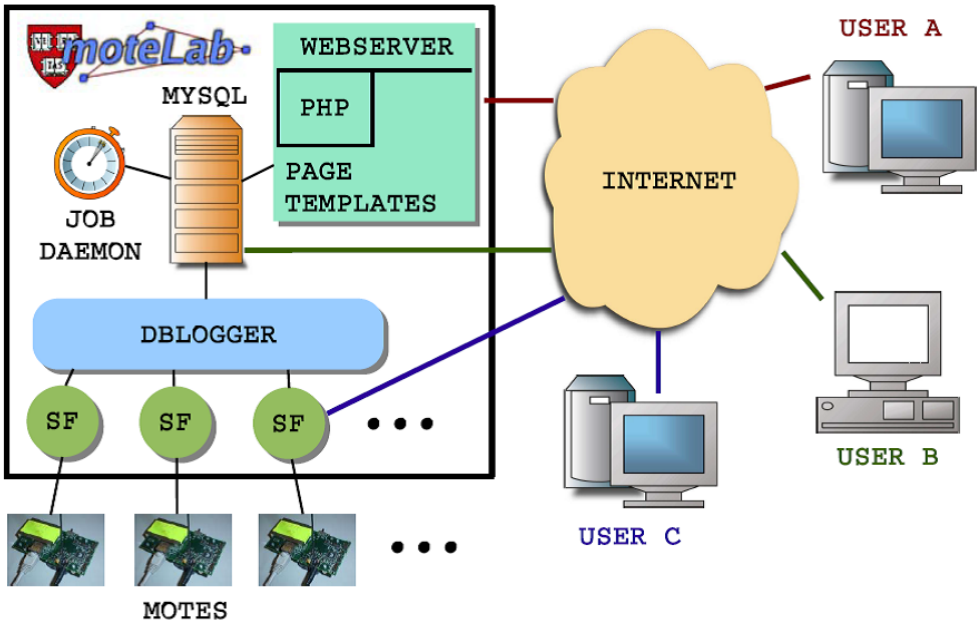


Figura 2.11: Arquitetura do MoteLab [53]

A figura 2.11 mostra os elementos de comunicação entre os diferentes componentes e uma ilustração duma atividade típica numa *testbed*.

Um tarefa de MoteLab consiste num determinado número de nós da *testbed* e executáveis, uma descrição que mapeia cada nó utilizado a um executável, vários ficheiros de classes Java utilizados para *logging* de dados e outros parâmetros de configuração. Para criar uma tarefa, um utilizador envia os executáveis necessários e as classes e descreve como os nós devem ser configurados. Uma vez a tarefa criada, o Motelab guarda a informação de configuração, permitindo que a mesma tarefa possa correr múltiplas vezes, com diferentes durações e/ou em diferentes partes do dia [53].

2.4.3 TARWIS

O TARWIS [57] é um sistema de gestão de *testbed* flexível e genérico, e corresponde a um sistema reutilizável em projetos de investigação ou educacionais de redes *wireless* de sensores. Este sistema oferece serviços essenciais para o funcionamento da *testbed*, como o acesso a multiplos utilizadores aos recursos da *testbed*, reserva *online*, configuração e agendamento de experiências, aquisição de dados automatizada, *logging* e observação em tempo-real de experiências. O TARWIS fornece estas funcionalidades independentemente do tipo ou do sistema operativo do nó. Este sistema foi projetado para permitir aos investigadores aceder

ao recurso das *testbeds* remotamente, pela Internet.

A arquitetura do TARWIS é demonstrada na figura 2.12. O *portal server* corresponde ao local onde as partes essenciais do TARWIS estão alojadas. Para além do componente de servidor do TARWIS (canto inferior esquerdo), o *portal server* aloja a interface *web* do TARWIS, que é protegida pelo sistema de autenticação e autorização Shibboleth [58]. De seguida, serão descritos os diferentes componentes indicados na figura.

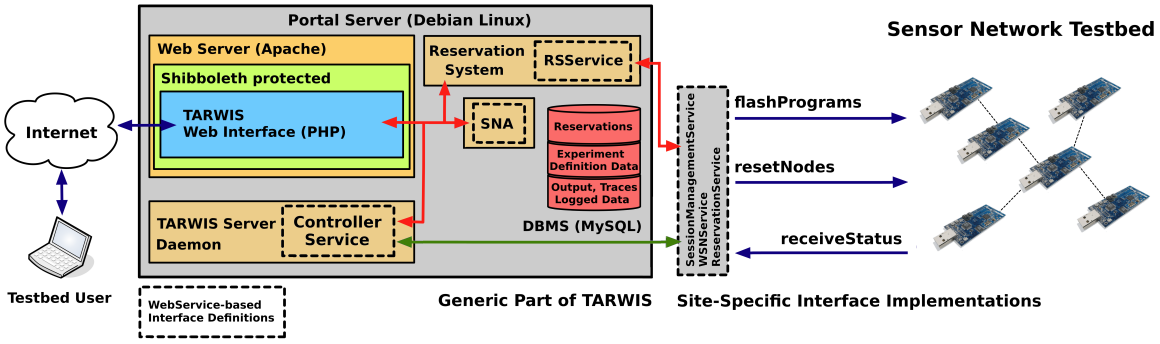


Figura 2.12: Arquitetura do TARWIS [57]

- *Portal server*: corresponde a um PC comum com um mínimo de 2GB de RAM e ligação banda larga à Internet. O TARWIS é fornecido com *scripts* de instalação para simplificar a configuração que foi testada e otimizada para Linux Debian Lenny [59].

- Interface *web* TARWIS: uma interface de utilizador simples, implementada como uma página *web* dinâmica em PHP, que oferece ao utilizador acesso à *testbed* via *browser web*. Um *DataBase Management System* (DBMS) MySQL é utilizado para guardar as configurações pessoais e dados das definições de experiências. Para aceder a esta interface, os utilizadores têm que proceder ao *login*, utilizando as credenciais do Shibboleth. Os utilizadores normais têm apenas a capacidade de enviar experiências, enquanto os administradores têm capacidade de adicionar nós extra, agendar tarefas de manutenção, apagar experiências de utilizadores normais, etc. Os visitantes só podem observar experiências declaradas como públicas.

- Shibboleth: a arquitetura TARWIS separa as preocupações de autenticação e autorização dos recursos. O Shibboleth é um sistema de autenticação/autorização de código aberto utilizado para *Single-Sign On* (SSO) *web* e gestão de contas de utilizador. Com o uso deste sistema, os utilizadores apenas carecem de uma conta com credenciais utilizador/*password* para efetuar *login* em todas as instalações do TARWIS.

- TARWIS *server daemon*: este é o processo principal que controla toda a execução de experiências. Este *daemon* encapsula o acesso à base de dados e todas as operações lógicas definidas nos ficheiros *Web Services Description Language* (WSDL). O TARWIS *server daemon* é inteiramente escrito em Perl. Este *daemon* arranca um subprocesso no momento em que uma nova experiência é agendada. Este subprocesso encontra a descrição da experiência na base de dados, verificando de seguida se a experiência submetida já tem uma reserva válida através do TARWIS *reservation system*. Este *reservation system* determina quais os nós que foram reprogramados e com que imagem de código binário. Depois, liga-se ao nós de sensores, através das definições de interface WSDL definidas no *reservation system*, *session manager* e WSN *service Application Programming Interface* (API), de modo a reiniciar e reprogramá-los. Sobre esta API, ao utilizador de *testbed* é proporcionada mais tarde a oportunidade de aceder e programar os nós de sensores, através da interface *web* TARWIS, durante a execução da experiência. Por fim, o TARWIS *server daemon* e os seus subprocessos recuperam os dados das experiências dos nós de sensores que são de seguida armazenados na base de dados.

- APIs/interfaces baseadas em *web services*: os componentes do TARWIS comunicam entre si sobre *web services*, usando descrições de API disponibilizadas no formato WSDL. As descrições das interfaces dos *web services* são independentes de linguagem de programação e de sistema operativo. Existem *bindings* de *web services*/*Simple Object Access Protocol* (SOAP) para as principais linguagens de programação e todos os principais sistemas operativos. Na figura 2.12, são exibidos três exemplos das funções da API que são invocadas pelo TARWIS: reprogramação de um nó, reinício de nós e recolha do *output* ou mensagens de estado de um nó. A implementação destes serviços pode residir na mesma máquina que os componentes do TARWIS, ou em qualquer outra máquina acessível a partir do *portal server*.

- TARWIS *resource reservation system*: este componente é utilizado para prevenir o acesso concorrente aos recursos da *testbed*. As reservas podem ser feitas para a totalidade de uma *testbed*, ou um subconjunto de recursos da mesma. O TARWIS *reservation system* expõe as suas primitivas para a interface *web* TARWIS, onde podem ser manipuladas através de um *browser*. Através do *web service* RSService, estas também podem ser manipuladas doutras formas, como usando um *script*.

- Serviço TARWIS *Sensor Network Athentication* (SNA): Cada instalação do TARWIS

dispõe de um *web service* para autorização. O processo de autorização, que fornece os direitos a utilizadores autenticados, é feito localmente para cada *testbed* TARWIS. Cada instituição que detém uma *testbed* pode especificar quais os direitos de acesso para um utilizador específico. Para assegurar modularidade e consistência, o serviço SNA está presente como um *web service* no *portal server*, podendo, adicionalmente, ser consultado por qualquer cliente que suporte *web services*, como, por exemplo, *scripts* em Perl ou Python [60] [57].

2.4.4 DES-TBMS

O *Distributed Embedded Systems Testbed Management System* (DES-TBMS) [61] é utilizado para facilitar o processo de experimentação para os investigadores, para a automação em redes *mesh wireless* e redes de sensores *wireless*. Este sistema de gestão é constituído por vários componentes, cada um deles responsável por tarefas específicas no processo de experimentação, incluindo:

- definição de descrições de experiências num formato padrão;
- agendamento e gestão de experiências;
- execução automática de experiências;
- monitorização distribuída dos nós da rede durante uma experiência;
- uma ferramenta de visualização para mostrar o estado da *testbed*;
- uma ferramenta de avaliação para determinar métricas de desempenho dos dados recolhidos de experiências.

O DES-TBMS utiliza uma linguagem de domínio específico para a definição de experiências, o DES-Cript. Como gestor de experiências utiliza o componente DES-Exp, que mantém todas as experiências guardadas na base de dados "Experiment Description" e é responsável pelo agendamento, execução e recolha dos ficheiros de *log* de uma experiência. Os resultados destas experiências são guardados na base de dados "Result Data". O DES-Mon é responsável pela monitorização dos nós durante as experiências, guardando os seus dados numa base de dados com o nome de "Monitoring Data". A visualização do estado da *testbed* é feita com o componente DES-Vis. Por fim, o DES-Val utiliza os dados recolhidos nas bases de dados

"Result Data" e "Monitoring Data" para determinar as métricas de desempenho para uma experiência em particular, conforme definido na sua descrição. Esta arquitetura é demonstrada no esquema 2.13 e os diferentes componentes são depois explicados com maior detalhe.

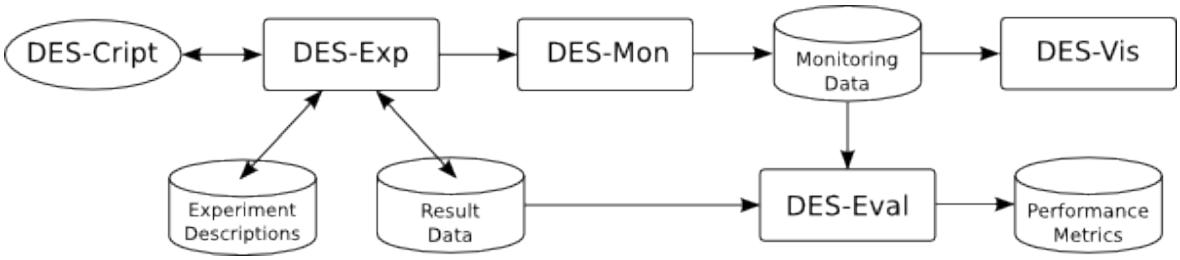


Figura 2.13: Arquitetura do DES-TBMS [61]

O DES-Exp fornece um gestor de experiências que controla as tarefas de criar, programar e executar experiências. A execução efetiva de uma experiência é realizada automaticamente: o utilizador especifica o tempo de início pretendido e é notificado por e-mail após a execução, quando os resultados estiverem disponíveis. Portanto, a realização da experiência não pode ser influenciada pelas interações entre os investigadores. Isto assegura um grau elevado de reprodutibilidade nos casos em que a mesma experiência é repetida várias vezes.

A funcionalidade do DES-Exp é fornecida por duas aplicações principais. A primeira é uma aplicação *web* implementada como um *servlet* Java, que permite ao utilizador criar novas experiências a partir do zero ou modificar experiências previamente definidas. Como a hora de início é escolhida pelo utilizador, é possível a execução automática de experiências durante a noite, quando são esperadas menos interferências causadas por possíveis redes paralelas. Além disso, uma unidade de execução consulta a base de dados, em intervalos regulares, para verificar se existem experiências pendentes, e, em caso de existirem, o DES-Exp reúne o arquivo de descrição associado à experiência e começa a execução. A unidade de execução usa uma API *Secure Shell* (SSH) para comunicação entre os nós. No início de uma experiência, são estabelecidas ligações SSH entre todos os nós participantes, que são utilizadas ao longo da experiência para configurar os nós e enviar comandos. Após a execução da experiência, todos os ficheiros de saída gerados por comandos são recolhidos para posterior processamento pelo DES-Eval. Após a pós-configuração da *testbed*, esta é libertada e podem assim ser efetuadas outas experiências adicionais. Quanto ao DES-Mon, a sua tarefa é monitorizar o estado da *testbed* durante a execução de uma experiência. É recolhido o estado dos nós participantes numa experiência, por pesquisa nesses mesmos nós, num intervalo fixo, conforme especificado no ficheiro de descrição da experiência, usando *Simple Network Management*

Protocol (SNMP) [62] [61]. Os dados recolhidos de cada verificação consistem no estado do *router mesh*, o estado das suas interfaces de WLAN e a tabela de *routing* do *kernel* de Linux. Esta última é utilizada para visualizar a conectividade durante a experiência com o DES-Vis. Usando a funcionalidade SNMP-Set, podem ser aplicados ajustes personalizados aos *router mesh* na fase de configuração. O DES-Mon é implementado como uma aplicação Java e usa um *framework* SNMP4J [63] [61].

O componente de avaliação DES-Eval analisa a saída gerada pelas ações de uma experiência e insere os resultados formatados na base de dados "Performance Metrics". Portanto, o DES-Eval automatiza a tarefa de recolher manualmente todos os arquivos de *log* e extrair as informações importantes para análise posterior. Ele fornece um conjunto de *scripts* em Python, para avaliação automática de ferramentas comumente utilizadas em experiências de rede, como ping, ttcp e iperf. O DES-Eval executa um *script* de avaliação que processa o arquivo de saída e fornece os resultados formatados. Os dados estatísticos de uma experiência, como média, desvio-padrão, variância e intervalos de confiança são utilizados para calcular as métricas de desempenho específicas. Num segundo passo, *scripts* adicionais podem ser utilizados para gerar gráficos padrão dos dados.

O DES-Vis é um software de visualização 3D baseado no *framework* JavaView [64], cujo principal objetivo é mostrar a conectividade da rede no decorrer das experiências e que se baseia em informações recolhidas na tabela de *routing* do *kernel*, durante as pesquisas de rede pelo DES-Mon. As mudanças na conectividade podem ser animadas ao longo de uma experiência. Para isso, as imagens dos estados de rede correspondentes são classificadas de acordo com o seu *timestamp* e progridem através deles. Este recurso permite aos investigadores descobrir rapidamente explicações para os resultados obtidos e simplifica conhecimento dos fenómenos observados.

O DES-Cript não corresponde propriamente a um componente, mas sim a uma linguagem de domínio específico para definir e descrever as experiências em rede, que podem ser executadas numa *testbed*. Esta linguagem visa simplificar as experiências ao usar um esquema de *design* compreensível e simples, garantindo a reprodutibilidade de experiências.

O DES-Cript foi desenvolvido com base em XML, herdando a sua propriedade de fácil leitura. Isso permite uma estrutura clara e hierárquica, tornando a criação de descrições de experiências compreensível. A estrutura da descrição está estreitamente mapeada para o fluxo de trabalho de uma experiência de rede. Todas as etapas consecutivas no fluxo de trabalho da experiência são representadas como secções diferentes nos ficheiros de DES-cript, que são

basicamente a configuração, execução e avaliação [61].

2.4.5 OMF

O *cOntrol and Management Framework* (OMF) [50] é um conjunto de componentes de *software* que fornece a gestão, controlo, ferramentas de medição e serviços para utilizadores e operadores de *testbeds* de redes. Este sistema de gestão foi originalmente desenvolvido para a *testbed wireless* ORBIT [65], ou seja, projetada para nós estáticos com dispositivos *wireless* 802.11, no Winlab [66]. Através do desenvolvimento ativo e extensões no NICTA [67], acabou por evoluir para um *framework* de código aberto, que suporta um número elevado de recursos heterogéneos com e sem fios.

O OMF é principalmente escrito em Ruby [68] e tem como objetivos fornecer um *framework* para *testbeds* unificado que responda aos desafios mencionados inicialmente. Atualmente, o OMF fornece um suporte significativo ao ciclo de experiências e os mecanismos para facilitar o suporte de uma federação global de *testbeds* e partilha de recursos. Este *framework*, de um modo abrangente, permite:

- Executar e recolher resultados de experiências;
- Fornecer um conjunto de serviços para a gestão e operação eficiente dos recursos duma *testbed*.

O OMF não está restrito a tecnologia específica da *testbed* e possibilita, a um utilizador, fazer uma descrição de alto nível de experiências, com o auxílio da linguagem *OMF Experiment Description Language* (OEDL) [50][49].

Pode ser encontrada no capítulo 3 uma explicação mais detalhada deste sistema de gestão, uma vez ter sido este o escolhido para este projeto.

2.4.6 Sumário e Conclusões

Relativamente aos três primeiros sistemas de gestão de *testbed* descritos (JAMES, Motelab e TARWIS), estes foram excluídos das hipóteses de sistemas de gestão para a *testbed* do DRIVE-IN por serem apenas apropriados para *testbeds* de redes *wireless* de sensores, embora forneçam uma gama de funcionalidades apropriadas aos requisitos inicialmente indicados. Assim, nenhum destes sistemas de gestão pode ser adequado à *testbed* do DRIVE-IN, pois esta corresponde a uma rede veicular que utiliza a norma IEEE 802.11p. Para além disso,

estes sistemas estão, atualmente, pouco acessíveis ao utilizador comum e/ou não se encontram em código aberto.

Relativamente ao DES-TBMS e o OMF, ambos estão disponíveis livremente em código aberto, e ambos correspondem a opções viáveis como sistemas de gestão da *testbed* do DRIVE-IN, até porque são semelhantes em termos da gama de funcionalidades [69]. O DES-TBMS fornece um tipo de sistema com uma interface para gerir experiências e ferramentas para monitorizar resultados. No entanto, o OMF apresenta melhores capacidades, com um método mais dinâmico e escalável. Outro fator a ter em conta é a linguagem, pois o DES-Cript é mais difícil de implementar e menos escalável na criação de experiências, quando comparado com a OEDL do OMF.

Assim, a versatilidade e a facilidade de implementação do OMF levaram à sua escolha para a aplicação na *testbed* do DRIVE-IN.

2.5 Conclusão

Ao longo deste capítulo foram apresentadas as tecnologias existentes no domínio das redes veiculares, assim como as suas características, explicando deste modo o que são as redes veiculares e como funcionam. De seguida, foram descritas as suas aplicações e objetivos, nomeadamente ao nível da segurança, otimização de tráfego e aplicações de conforto.

Por fim, foram apresentadas as características a ter em conta na escolha de um sistema de gestão de *testbed*, tendo sido descrito algum do *software* existente nesta área. Pode-se concluir que existem vários sistemas de gestão de *testbed* que respondem aos objetivos propostos para a aplicação na rede veicular do DRIVE-IN; no entanto não existe nenhum sistema apropriado especificamente para as redes veiculares. Portanto, além da aplicação de um destes sistemas na *testbed* do DRIVE-IN, é ainda necessário proceder a alterações para a sua aplicabilidade na rede veicular. Tendo-se escolhido o OMF para esta implementação, pode-se encontrar no capítulo 3 uma descrição mais detalhada do mesmo. O capítulo 4 apresentará a implementação que foi realizada para adequar este sistema à *testbed* do DRIVE-IN.

Capítulo 3

Sistema de Gestão da *Testbed*

3.1 Introdução

Para o cumprimento dos objetivos propostos, como já foi referido, foi necessário escolher um sistema de gestão para integrar na *testbed*: este foi o OMF. Este *framework* genérico para a gestão de *testbeds* de redes permite fazer uma descrição sistemática e instrumentação das experiências, oferecendo as ferramentas para a execução automática das mesmas, controlo e obtenção das medições que provêm delas (figura 3.1) [50]. Neste capítulo é dada uma explicação mais detalhada da arquitetura deste sistema de gestão, bem como dos seus componentes e protocolos.

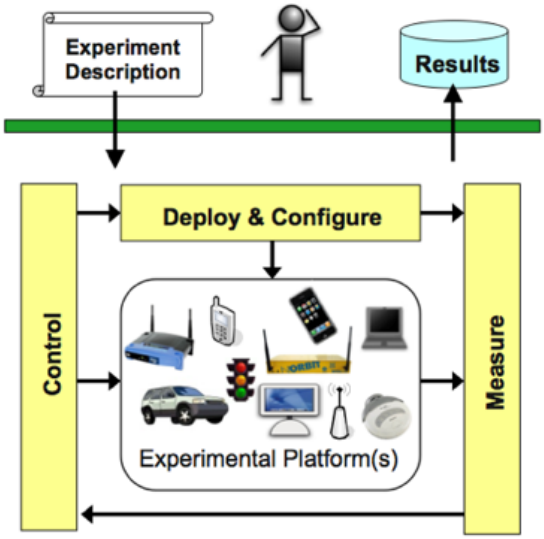


Figura 3.1: *Overview* do funcionamento do OMF [50]

3.2 Componentes do OMF

A arquitetura do OMF assenta em três componentes: um *Aggregate Manager* (AM), um *Experiment Controller* (EC) e múltiplos *Resource Controllers* (RC). Também existem outros dois componentes associados ao sistema de gestão que é necessário referir: os *Experiment Description* (ED) e o *OMF Measurement Library* (OML). As definições e as funções destes componentes são descritas nas subsecções seguintes.

3.2.1 Aggregate Manager

O AM é uma coleção de serviços que tem como funções gerir todos os recursos duma *testbed*, fornecer informação sobre os recursos disponíveis (através de uma base de dados) e relacionada com a *testbed* e arrancar e parar *software*, que distribui imagens de disco para dispositivos de armazenamento destes recursos. O AM também tem a utilidade de controlar o *boot* de um recurso (da unidade de armazenamento ou de uma imagem através de *Preboot eXecution Environment* (PXE)) e fornecer o acesso às bases de dados com os dados recolhidos. Para o efeito, o AM utiliza uma série de serviços que são fornecidos pelos próprios servidores *Hypertext Transfer Protocol* (HTTP) e *eXtensible Messaging and Presence Protocol* (XMPP) [50].

3.2.2 Experiment Controller

O EC opera com o AM da *testbed* envolvido numa experiência e é responsável pela execução, controlo e instrumentação da *testbed*. O utilizador fornece uma descrição da experiência em OEDL ao EC que, por sua vez, comunica com o AM da *testbed* para configurar os nós de acordo com a descrição. Seguidamente o EC envia comandos, de acordo com a descrição do utilizador, aos RC envolvidos na experiência, que procederão à sua realização. Podem existir vários EC em simultâneo, cada um responsável pela sua experiência. É com o EC que os utilizadores da *testbed* interagem diretamente [50].

3.2.3 Resource Controller

O RC está presente nos recursos (ou nós) da *testbed*, sendo responsável por escutar os comandos recebidos pelo EC, executá-los e enviar as respostas de volta, com a informação resultante desses comandos. Também é o RC o responsável pela tarefa de receber e escrever as imagens de disco na unidade de armazenamento do recurso [50].

3.2.4 Experiment Description

Um ED é um *script* onde é descrita uma experiência pelo utilizador que vai correr na *testbed*, sendo esta escrita na linguagem de domínio específico do OMF, o OEDL. Quando é descrita uma experiência nesta linguagem, a mesma é transferida para o *framework* que, de seguida, irá configurar os nós da *testbed*, fazer o arranque de aplicações e, durante o decorrer da experiência, recolher os dados delas de acordo com a descrição do utilizador. As descrições das experiências nesta linguagem podem ser divididas em duas partes:

- Configurações e requisitos dos recursos: onde se enumeram os diferentes recursos que vão ser utilizados numa experiência juntamente com as suas configurações, aplicações que vão ser utilizadas, também com as respetivas configurações e medições a ser retiradas.
- Descrição de tarefas: onde se descreve a sequência da experiência naquilo que é, basicamente, uma máquina de estados onde são descritas as diferentes tarefas que a experiência vai realizar nos diferentes recursos, e onde é definindo também o tempo de duração das diferentes partes da experiência, ou então a descrição de eventos que vão ser despoletados pelas medições obtidas [50].

3.2.5 OML

A *Measurement Library* (ML), ou, mais especificamente, o OML, é um conjunto de um servidor e cliente para a recolha de dados das medições das experiências. É o ML que permite a instrumentação das aplicações de onde se pretendem retirar os resultados. O OML permite uma recolha de dados em tempo real de vários *Measurement Point* (MP) durante o decorrer de uma experiência. O MP é uma parte dentro de uma aplicação, onde um conjunto de métricas provenientes da mesma podem ser recolhidas pelos instrumentos de medição do OML. Um MP pode ser criado, por exemplo, num gerador de tráfego, de modo a recolher o tamanho do pacote, *sequence number* ou a *bit rate* [50].

3.3 Arquitetura do OMF

Segundo exemplificado na figura 3.2 o processo de execução de uma experiência no OMF desenrola-se da seguinte forma:

1. Primeiro é arrancado o EC, passando para este a descrição da experiência em linguagem OEDL;

2. O EC, após interpretação da descrição fornecida, acede aos serviços do AM da *testbed*, pedindo-lhe a inicialização e a configuração dos recursos necessários;

3. O AM procede à inicialização e configuração destes recursos;

4. Assim que todos os recursos estão preparados, o EC comunica com os RC presentes neles, enviando-lhes comandos;

5. Os RC interpretam e executam estes comandos podendo, entre várias possibilidades, arrancar uma determinada aplicação e alterar os parâmetros de uma que se encontre em execução;

6. Depois do passo anterior a experiência está no estado "running" e as aplicações, se assim definidas, podem enviar dados de medições para o ML;

7. O ML pode fazer algum pré-processamento aos dados recebidos, como por exemplo, cálculo de médias, máximos ou mínimos;

8. Numa fase seguinte o ML envia os dados recolhidos para o *Measurement Collection Service* (MCS) (o servidor OML), que guarda os guarda numa base de dados única e exclusiva para esta experiência.

Conforme é apresentado na figura 3.2, é possível correr mais do que uma experiência em simultâneo, embora não seja suportado o uso dos mesmos recursos em mais do que uma experiência. Também é necessário garantir que as várias experiências não interferem entre si, tendo o cuidado de configurar, por exemplo, canais diferentes quando se usa uma interface *wireless* [50].

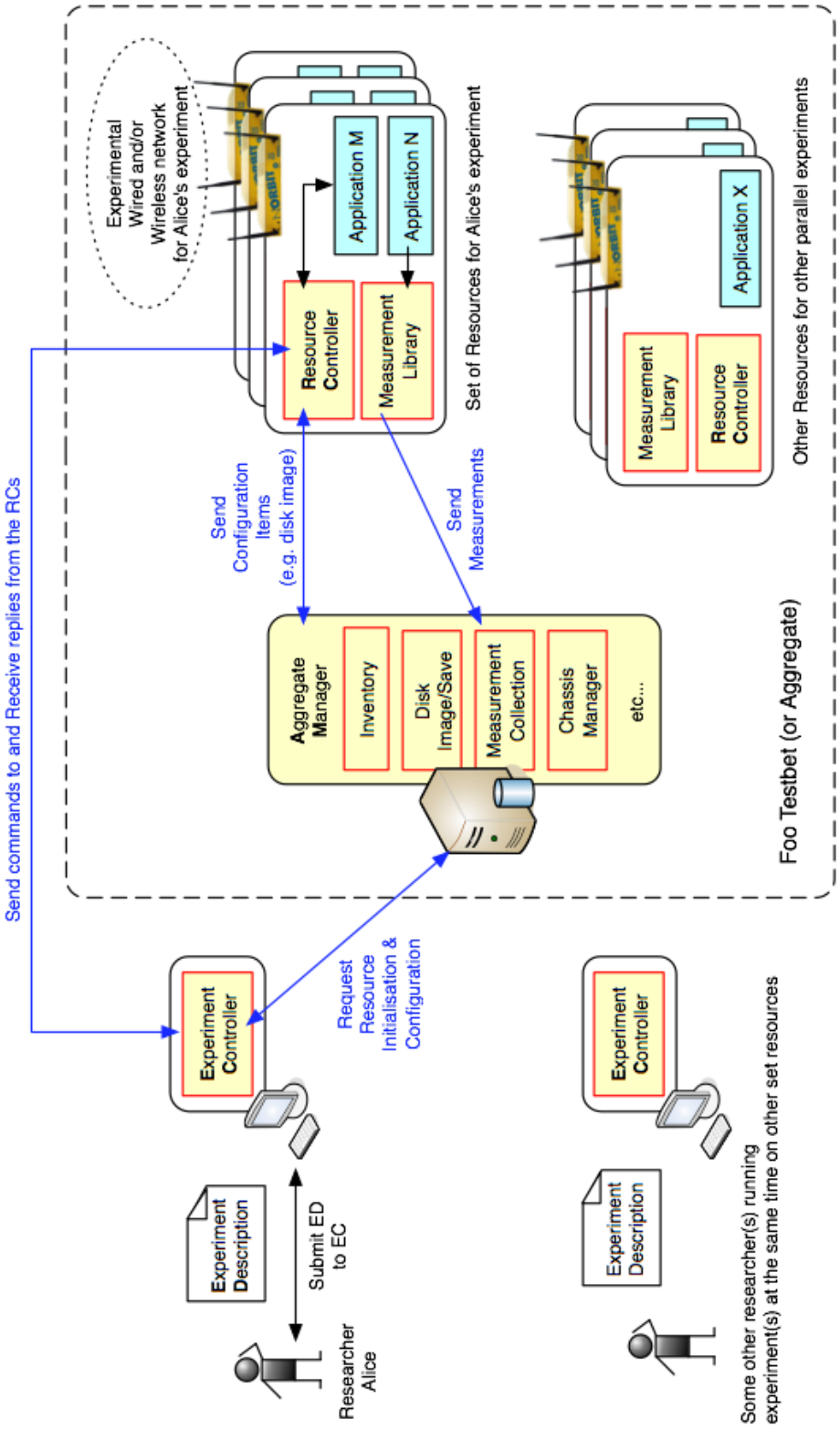


Figura 3.2: Arquitetura do OMF [50]

3.4 Arquitetura do OML

O OML baseia-se numa arquitetura cliente/servidor, sendo estes:

- OML *Measurement Library*: o cliente, responsável por retirar as medições dentro da experiência;
- OML *Collection Server*: o servidor, responsável por guardar os resultados das medições numa base de dados.

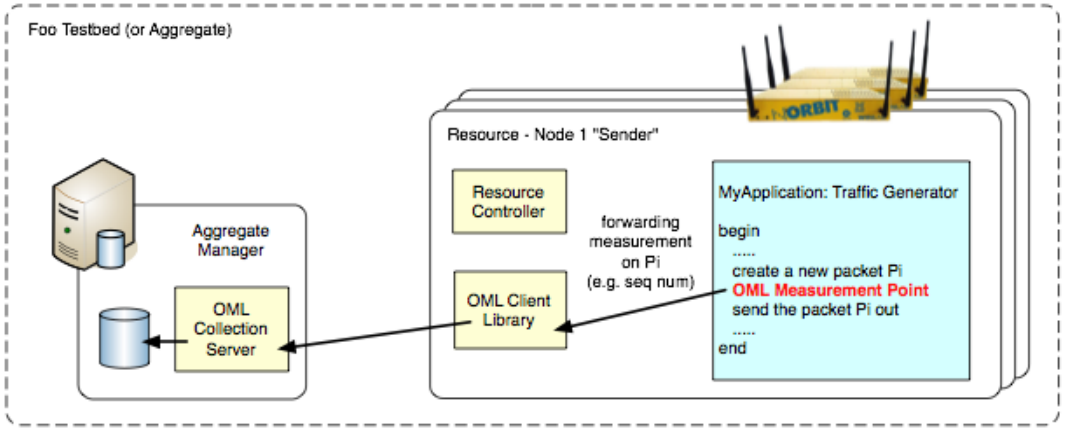


Figura 3.3: Arquitetura do OML [50]

O *Collection Server* corre como um *daemon* no servidor da *testbed* e recolhe os resultados dos vários recursos especificados nas experiências, guardando-os numa base de dados *sqlite* [70], única e exclusiva para cada experiência. O *Measurement Library* está disponível como uma biblioteca em cada um dos recursos utilizados nas experiências que é incluida nas aplicações. Desta forma, as aplicações enviam os dados das medições para o cliente do OML, que por sua vez pode aplicar algum pré-processamento aos dados e envia-os para o servidor, o *Collection Server*. As configurações das aplicações que usam o OML podem ser fornecidas através de parâmetros de entrada das próprias aplicações, ou então usando um ficheiro de configuração XML. As configurações que podem ser efetuadas são a identificação da experiência, a identificação da instância aplicação, o número de envios por segundo das medições para o servidor, o *Uniform Resource Identifier* (URI) do servidor OML, entre outros. Uma vez terminada a experiência, os resultados podem ser acedidos pelos utilizadores na referida base de dados *sqlite*. A figura 3.3 ilustra esta arquitetura na sua integração com o OMF.

O OML permite ainda a execução de experiências em modo *disconnected*. Para o efeito existe um servidor *proxy* que é configurado para guardar a informação recolhida das medições das experiências, e mais tarde, configurado de novo para as enviar para o servidor [50][71].

3.5 Protocolos

Esta secção descreve os protocolos utilizados pelo OMF na comunicação entre os vários elementos.

3.5.1 PXE

O carregamento das imagens de disco é feito através do protocolo PXE [72]. Este protocolo permite a um sistema fazer um *network boot*, ao tornar uma interface de rede num dispositivo de *boot*. O PXE arranca o sistema através da rede transferindo um ficheiro *boot image* a partir de um servidor, podendo este ser qualquer sistema operativo ou um agente "pre-OS" (um pequeno ambiente operacional para realizar alguma tarefa antes de ser carregado o sistema operativo final). Para a utilização do PXE são necessários alguns componentes no servidor, além do servidor PXE, como um servidor *Trivial File Transfer Protocol* (TFTP) e um servidor *Dynamic Host Configuration Protocol* (DHCP) (podendo este ser o mesmo que o servidor PXE). O servidor PXE espera pelos *discovery requests* do DHCP que incluem uma *tag* que identifica o cliente como um cliente PXE. Neste caso, o servidor PXE, responde ao cliente com a informação de configuração, incluindo o nome de um ficheiro de imagem para o *boot*. A imagem é então transferida usando o TFTP e, seguidamente, utilizada para fazer o *boot* do cliente [73]. Existem uma série de dispositivos de rede que suportam o PXE; no entanto, a realidade da implementação presente impossibilita o seu uso, como será descrito na secção 4.5.

3.5.2 XMPP PubSub

O OMF usa o protocolo XMPP para a comunicação entre o EC e os diferentes nós que contêm os RC, utilizando o método de PubSub (*Publisher/Subscriber*) [74] [50]. Todas as entidades registam-se no servidor XMPP e subscrevem-se a um conjunto de nós PubSub. As mensagens são publicadas para um determinado nó PubSub e são, então, retransmitidas para todos os nós subscritos pelo servidor XMPP, estando estas mensagens no formato XML.

No caso do OMF, as mensagens podem ser enviadas tanto para um nó específico como

para grupos. Um grupo pode ser um grupo de nós conforme definido num ED, ou então um grupo de todos os nós de uma experiência, sessão ou domínio. Cada nó, grupo, experiência, sessão e domínio é representado por um nó PubSub. Desta forma, os membros de um grupo subscrevem-se ao seu nó PubSub do grupo, os membros de uma experiência subscrevem-se ao nó PubSub dessa experiência e cada nó subscreve-se ao seu próprio nó PubSub, que tem o seu identificador ou endereço IP. De forma a enviar mensagens para um determinado nó ou grupo, é apenas necessário escolher o nó PubSub para o qual publicar.

Alguns dos nós PubSub são estáticos, os quais são tipicamente criados pelo AM; todos os outros estão relacionados com as experiências ou sessões ativas, os quais são criados pelo EC, que apaga os nós no término da experiência. O RC não cria nenhum nó PubSub por si próprio, apenas subscreve aos nós criados pelo AM e EC [50].

3.6 Desafios e Conclusão

Neste capítulo, foi apresentado mais detalhadamente o OMF, o sistema de gestão escolhido para aplicação na *testbed* do DRIVE-IN. No presente capítulo foram descritos os seus componentes, a sua arquitetura e os protocolos associados, bem como uma explicação da arquitetura do OML. Pode-se concluir que este sistema de gestão fornece as funcionalidades necessárias para o cumprimento dos objetivos propostos para esta Dissertação. No entanto, após análise, conclui-se que, mesmo com a instalação deste sistema, é ainda necessário proceder a alterações por forma a estender a sua aplicabilidade à *testbed* específica a ser utilizada pelo DRIVE-IN. Os problemas que surgiram foram os seguintes:

- A existência de nós móveis impossibilita o uso de ligações cabladas, inviabilizando a comunicação do EC com os RC. No sistema a ser desenvolvido serão utilizadas de placas de banda larga móvel;
- O funcionamento do PXE depende da ligação cablada, portanto foi necessário encontrar uma alternativa para o carregamento de imagens de disco. No sistema a ser desenvolvido esta alternativa faz uso da rede 3G;
- O tamanho das imagens de disco a serem carregadas nos nós deve ter um tamanho reduzido para que possam ser enviadas pela rede móvel;
- A ligação 3G pode falhar e é necessário encontrar uma solução para esta situação. No sistema desenvolvido será atualizado o endereço IP dinamicamente na base de dados do

AM;

- A rede de experiências no caso do DRIVE-IN usa a norma IEEE 802.11p, para o qual o OMF não está preparado. O sistema desenvolvido foi estendido para suportar o IEEE 802.11p;

- A ligação à Internet nos táxis deve ser partilhada para usufruto dos motoristas e clientes da RadiTáxis, portanto foi necessário encontrar uma maneira de gerir os acessos a esta ligação, obtendo a informação dos acessos feitos à Internet efetuados e a sua duração.

As respostas a estes desafios e a descrição da implementação do sistema de gestão da *testebed* são apresentadas no capítulo 4.

Capítulo 4

Implementação do Sistema de Gestão

4.1 Introdução

Neste capítulo é apresentada uma explicação mais detalhada da forma como a implementação do sistema de gestão foi realizada, para a sua integração na *testbed* do DRIVE-IN. São de seguida enumeradas as diferentes secções deste capítulo, em conjunto com a descrição do seu conteúdo:

- Na secção "Arquitetura da Implementação" pode-se encontrar a solução alcançada para a arquitetura de instalação do OMF na referida *testbed*, juntamente com a comparação com a instalação numa *testbed* de redes comum, bem como a descrição do material utilizado para integrar nos nós da *testbed* (conforme foi utilizado para a instalação);

- Em "Instalação do OMF"é descrita a forma como o sistema de gestão foi instalado e configurado nos nós da *testbed* e no servidor;

- Na secção "Extensão de Funcionalidades" são descritas as novas funcionalidades incluídas no OMF e a forma como foram implementadas, bem como as motivações para o seu desenvolvimento;

- Seguidamente, em "Método de Recolha de Resultados" é explicada a forma como os dados de medições resultantes das experiências são obtidos a partir dos diferentes nós da *testbed* do DRIVE-IN;

- Em "Imagem das Placas" é fornecida toda a informação sobre a geração de imagens de disco para o funcionamento com o OMF e a sua motivação, assim como todos os processos que estão relacionados, como as aplicações instaladas nas placas;

- Por fim, pode ser contemplada na última secção deste capítulo, "Provimento de Internet nos Táxis", a forma como foi respondido o problema que implica o fornecimento dos serviços Internet nos táxis, a qual vai ser gerida por esta plataforma de gestão da *testbed*.

4.2 Arquitetura da Implementação

Na implementação do sistema de gestão OMF na rede veicular do DRIVE-IN foi necessário encontrar uma nova arquitetura (figura 4.2), pois a instalação do sistema conforme sugerida pela equipa do OMF (figura 4.1) não se aplica na rede veicular em questão. Na instalação oficial pode-se encontrar uma rede cablada para controlo (azul na figura), que liga o AM e o EC a um *switch*, que por sua vez liga aos vários nós da *testbed* onde estão os RC. Estes nós estão ligados também numa rede de experiências, que pode ser *wireless* ou cablada (vermelho e verde na figura, respetivamente). O EC também está numa máquina separada do AM, o que não é necessário, mas também não é inválido na aplicação à rede veicular.

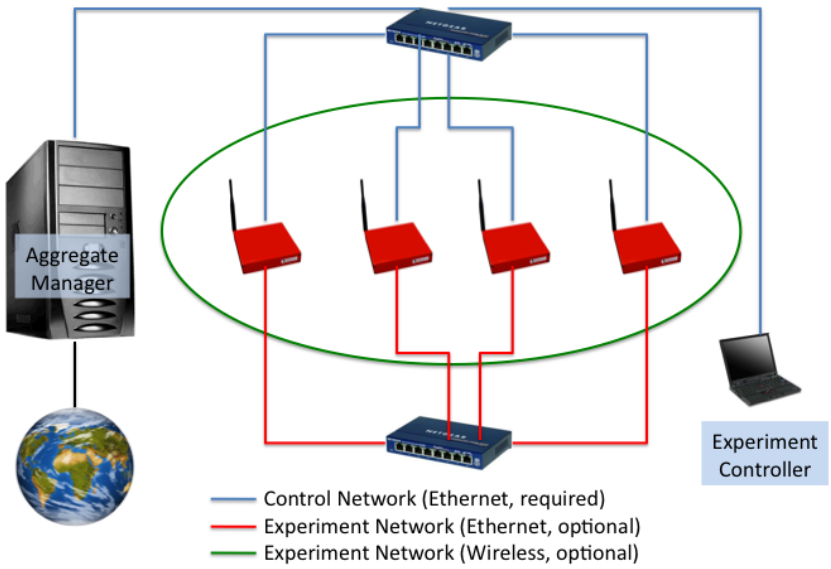


Figura 4.1: Diagrama de instalação proposto pelo OMF [50]

No caso da *testbed* do DRIVE-IN, toda a parte do AM e EC pode ser instalada conforme o sugerido; no entanto, ambos foram instalados na mesma máquina de servidor localizada no IT de Aveiro, por uma questão de simplificação. Com esta instalação é possível ter vários utilizadores a aceder em simultâneo a esta máquina e usar o EC simultaneamente. No caso dos RC a aplicação teve que ser diferente da sugerida no caso anterior. Como se pode ver na figura 4.2, a rede de controlo passou a funcionar através da Internet e, mais especificamente, no caso dos RC, através da rede celular 3G. A rede de experiências passou a funcionar exclusivamente através da rede veicular na norma IEEE 802.11p.

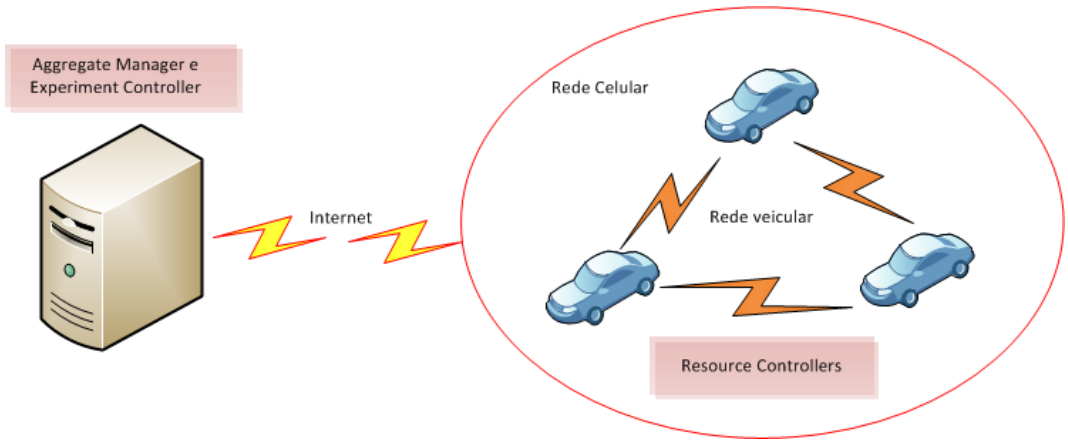


Figura 4.2: Implementação do OMF na *testbed* do DRIVE-IN

Segundo fornecido em [50], os requisitos das máquinas que devem integrar a instalação do OMF são os seguintes:

- Para o AM: um PC com duas interfaces *ethernet*, uma para controlo, outra para acesso à Internet;
- Uma série de nós da *testbed*, em que cada um destes deve ter uma porta *ethernet*, para controlo, e uma ou mais interfaces de rede, para as experiências. É requerido 32MB de memória RAM, sendo que o recomendado é 64MB ou mais;
- Um interpretador da linguagem Ruby [68], em que o OMF foi escrito, embora algumas ferramentas adicionais (como, por exemplo, para o carregamento e gravação de imagens de disco) necessitem de compilação, caso não se faça a instalação numa plataforma x86 ou x86_64, cujo binário é fornecido;
- Placas *wireless* da Atheros (driver *ath_pci*), Cisco (driver *airo_pci*) e Intel (driver *ipw*

2200) são suportadas pelo OMF, podendo ser adicionado um *wrapper* para outros drivers;

- Um *switch ethernet* para a rede de controlo, onde se devem ligar todas as interfaces de controlo do AM e dos nós;
- Se a rede de experiências usar *ethernet*, é necessário um outro *switch* (ou VLAN separadas) para ligar as respetivas interfaces.

Analizando estes requisitos, o da existência de duas interfaces *ethernet* no AM foi eliminado quando se tomou a decisão de usar uma interface de rede móvel para o controlo. Assim, o servidor onde o AM foi instalado necessitava apenas de ter uma ligação persistente à Internet. O mesmo acontece para as placas que compõem os nós da *testbed*, sendo que estas apenas contêm uma interface *ethernet*, mas que não foi de todo utilizada, pois a rede de controlo passou a ser 3G e a interface para as experiências é a da norma IEEE 802.11p. Embora esta interface use um *driver* Atheros, há que tomar em consideração que este foi alterado [75], o que obrigou a uma alteração do código original para o passar a suportar.

Considerando que a rede celular é utilizada para o controlo, toda a parte do OMF que inclui o requisito de uma rede de controlo cablada foi eliminado, sendo fulcral desenvolver um novo suporte para algumas funcionalidades, sobre a Internet, como é o caso do carregamento de imagens de disco.

Por fim, os dois últimos requisitos não se aplicam à implementação em questão, pelo facto de não ser utilizada *ethernet*.

O material utilizado para compor os diferentes nós da *testbed*, que devem ser integrados nos veículos da figura 4.2, é descrito de seguida:

- Módulo Alix3D3: um mini-PC com um CPU 500 MHz AMD Geode LX800, 256 MB DRR DRAM e slot para cartão *CompactFlash*;
- Módulo Wi-Fi preparado para a norma IEEE 802.11p/DSRC;
- Cartão de memória *CompactFlash* de 4 GB;
- USB *Modem* HSUPA, modelo MF636 da ZTE, preparado para a conetividade à rede celular 3G (figura 4.4);
- Módulo Wi-Fi USB preparado para a norma IEEE 802.11a/b/g/n, TP-Link modelo TL-WN722N (figura 4.3);

- Antena L-Com omnidirecional para frequências entre 5,150 e 5,9 GHz, da norma IEEE 802.11p.

Como pode ser visualizado na figura 4.5, os componentes foram preparados para a utilização e teste do sistema de gestão. Pode-se observar a placa Alix3D3, protegida por uma caixa de metal, com a antena L-Com, o módulo Wi-Fi USB TP-Link, utilizado na secção 4.8, e o *modem* USB. Na aplicação do equipamento num veículo é ainda necessário usar um módulo GPS, cuja necessidade principal é explicada na subsecção 4.5.2.

Figura 4.3: TPLink TL-WN722N [76]

Figura 4.4: TMN ZTE MF636 [77]

Figura 4.5: Placa para a *testbed* pronta a utilizar

4.3 Instalação do OMF

Tendo por base o equipamento descrito anteriormente, foi inicialmente executada uma instalação do OMF em três placas. Uma das placas funcionou como servidor, facultando os serviços de AM e EC, enquanto as duas outras placas, que eram idênticas, funcionaram como RC. Posteriormente, o AM e o EC foram transferidos para um servidor no IT, em Ubuntu 12.04 [78], e os RC foram incluidos numa *testbed* maior com 10 placas. Este processo desenrolou-se da seguinte forma, no servidor que contém o AM e o EC:

1. A instalação do Ruby, que é a linguagem de interpretação em que foi escrito o OMF, através do repositório do Ubuntu;

2. Instalação do servidor de XMPP recomendado, que corresponde ao Openfire 3.7.1 [79]. Este serviço XMPP é utilizado para a comunicação entre os RC e o EC, usando o método PubSub, conforme foi explicado no capítulo 3. O servidor XMPP foi configurado com o nome de domínio da máquina em que foi instalado ("vtestbed.nap.av.it.pt").

3. Instalação e configuração do AM, sendo esta executada através do repositório do OMF.

A seguir procedeu-se à instalação do OML (para a recolha dos dados das medições resultantes das experiências). No caso desta implementação, a configuração do serviço de PXE é desnecessária e são também incluídos os ficheiros que contêm as configurações para o serviço de suporte à aplicação de visualização de resultados do OMF. Cada um destes ficheiros de configuração corresponde a uma gama de serviços disponibilizados pelo OMF. Dentro destas gamas de serviço, cada uma fornece uma série de serviços de forma heterogénia. Por exemplo, o "inventory" fornece os serviços relacionados com a base de dados, utilizada pelo AM e com o mesmo nome, e o "frisbee" contém os serviços relacionados com as imagens de disco.

4. A base de dados que o OMF usa para a gestão dos recursos disponíveis é denominada de "inventory". Sendo uma base de dados MySQL [54], foram instalados os requisitos da mesma: *mysql-server*, *libdb4.6* e *phpMyAdmin* [80] e ainda o servidor HTTP Apache 2 [81], para suporte do *phpMyAdmin*. Foi criado o utilizador por defeito do OMF no *mysql-server*. A base de dados "inventory" foi concebida de seguida e foi inicialmente populada pelo *sample* fornecido para a instalação. Com o *phpMyAdmin* é possível aceder a esta base de dados através de uma interface *web*.

5. Para se permitir uma comunicação entre o servidor e as placas da *testbed* é necessário proceder à criação de nós PubSub no servidor XMPP [74] antes correr experiências. Primeiro tem que ser criada a árvore de nós PubSub do lado do sistema:

```
omf_create_psnode -5.4 vtestbed.nap.av.it.pt mksys
```

em que "vtestbed.nap.av.it.pt" é o nome de domínio do servidor XMPP. Os nós da *testbed* devem ser adicionados numa *slice*, sendo que esta separa os nós em diferentes blocos para diferentes funções. Por exemplo, a "pxe_slice" deveria ser utilizada para enviar e gravar imagens de disco nos/dos diferentes nós. Para se poderem usar os nós em experiências, estes têm que ser adicionados a uma *slice* com o seguinte comando:

```
omf_create_psnode -5.4 vtesbed.nap.av.it.pt mkslice
default_slice omf.my.drivein1 omf.my.drivein2
```

Desta forma, é criada uma *slice* no Openfire com o nome "default_slice", onde são adicionados os nós com os *Human Readable Name* (HRN) "omf.my.drivein1" e "omf.my.drivein2". Estes, no servidor XMPP Openfire, ficam registados como utilizadores com os nomes: "default_slice-omf.my.drivein1" e "default_slice-omf.my.drivein2". Esta *slice* é

utilizada para correr as experiências na *testbed*. É importante notar que, cada vez que se acrescentam novos nós na *testbed*, é necessário adicioná-los da mesma forma a uma *slice*, de modo a poderem ser utilizados em experiências. O *daemon* do AM deve ser reiniciado para que reconheça a nova configuração:

```
/etc/init.d/omf-aggmgr-5.4 restart
```

6. O EC foi também instalado na mesma máquina através do repositório do OMF. O nome de domínio do servidor XMPP foi colocado em ":communicator:xmpp:pubsub_gateway:". No campo ":web:host:" foi escrito o endereço de IP do servidor do IT. Por fim, foi colocado em ":omluri:", no formato "protocolo:hostname:porto", a informação de contacto do servidor OML.

7. Foi instalado o pacote de visualização de experiências do OMF, através do respetivo repositório.

Nas placas da *testbed*, não é possível instalar o RC a partir do repositório devido à imagem utilizada (secção 4.7). Portanto, foi feito o *download* do código fonte a partir do Git [82], do repositório do OMF, e foi feita a instalação nas placas. Como o OMF é essencialmente escrito em Ruby, bastou copiar os ficheiros para as respetivas pastas em que ficariam com uma instalação a partir do repositório para Ubuntu e proceder à sua configuração.

Em relação à base de dados do OMF, "inventory", é de notar que existe uma série de tabelas: "devices", "device_kinds", "device_ouis", "device_tags", "inventories", "location", "motherboards", "nodes", "pxeimages" e "testbeds". No entanto, no contexto da *testbed* do DRIVE-IN, atualmente, existe apenas a necessidade de usar as tabelas "nodes" e "testbeds". Na tabela "nodes", que contém informação dos nós da *testbed*, as colunas com relevância são: "control_ip", onde é guardado o IP da interface de controlo, e "hostname" e "hrn", que contêm o *hostname* e o HRN, respetivamente, de cada nó. Foi acrescentada a tabela "Carro", que irá conter a informação do veículo em que será instalada cada uma das placas da *testbed*, estando esta tabela ligada por uma chave estrangeira à tabela "node". Mais adiante, na secção 4.5 descreve-se o precosso de inserção de novas colunas na tabela "nodes": "LastPing", "imaging_progress" e "current_image", que auxiliam novas funcionalidades implementadas no sistema de gestão.

Cada nova entrada na base de dados deve ser inserida manualmente, existindo contudo algumas informações inseridas de forma dinâmica e automática, que serão referidas nas secções seguintes.

4.4 Instalação do *Modem* USB

Conforme é descrito nesta secção, os *modem* USB foram instalados nas placas. O primeiro desafio da instalação dos *modem* USB nas placas foi conseguir que estes fossem reconhecidos pelo sistema operativo Linux, pois não o são por defeito, apenas garantindo compatibilidade com algumas versões do sistema operativo Windows. Para o efeito foram utilizadas duas aplicações: *usb_modeswitch* [83] e o *daemon pppd*. O primeiro faz com que o *modem* seja reconhecido pelo sistema operativo, enquanto que o segundo tem a função de criar uma ligação *dial-up* à Internet através do dispositivo, usando o protocolo *Point-to-Point Protocol* (PPP).

Tornou-se também indispensável a configuração do *kernel* de Linux, com os módulos necessários ao reconhecimento do *modem*. Foram ativadas as opções: "USB Serial Converter Support" (como módulo), dentro desta opção foi ativado o "Generic Serial", o "Usb Driver GSm" (módulo) e "CDMA modems"; na opção "Network device support" foi ativado o suporte para PPP. Não especificamente, é essencial garantir que todas as opções das quais depende o funcionamento de dispositivos USB estão ativas: "Support for Host-side USB", "OHCI HCD support" e "USB Mass Storage Support".

A configuração do *usb_modeswitch*, para o reconhecimento do *modem* utilizado, é escrita no ficheiro "/etc/usb_modeswitch.conf". A aplicação deve então ser arrancada com o comando

```
usb_modeswitch -c /etc/usb_modeswitch.conf
```

de forma a que as configurações sejam lidas e a que o *modem* seja reconhecido pelo sistema operativo, criando os ficheiros do dispositivo USB na pasta "/dev/": "ttyUSB0", "ttyUSB1" e "ttyUSB2". As configurações para o modelo específico do *modem* são fornecidas, à partida, pela aplicação *usb_modeswitch*.

Com o *pppd* é necessário usar dois *scripts*: um com a configuração do *modem* e outro com a sequência de ligação. O primeiro, localizado na pasta "/etc/ppp/peers/" com o nome de "tmn3g", contém a configuração específica que define o dispositivo que se usa ("tyyUSB2"), o modo de autenticação para a rede da 3G, restabelecimento da ligação em caso de falha, entre outros, e uma chamada ao segundo *script*; o segundo, localizado na pasta "/etc/chatscripts/", com o nome de "tmn3g.chat", contém os comandos que são enviados ao dispositivo durante o processo de ligação à rede celular, incluindo o código PIN, *username* e *password* e o número telefónico da ligação *dial-up*. A configuração do código PIN foi colocada num ficheiro separado, de forma a facilitar a sua alteração, visto cada placa ter um *modem* com um cartão e PIN diferentes [84].

Após o término desta instalação, pode ser iniciada a ligação à Internet pela rede celular com o comando "pppd call tmn3g". Nesta fase, o sistema de gestão está pronto a utilizar a Internet como rede de controlo, no entanto ainda permanecem alguns problemas, cuja explicação é descrita na próxima secção.

4.5 Extensão de Funcionalidades

A ligação 3G nem sempre é estável e, mesmo havendo restabelecimento da ligação, que mantém a placa com ligação à Internet, o IP da interface do *modem* é dinâmico e atribuídos através de servidores DHCP, estando fora de controlo tê-lo de outra forma. Não tendo IP fixo, torna-se impossível ao OMF conhecer os novos endereços IP de controlo das placas acedendo à base de dados "inventory", pois manterá o IP que foi inicialmente inserido.

Outro desafio que surgiu está relacionado com a tolerância a falhas das placas da *testbed*. No sistema proposto é preciso garantir que, quando uma placa fica bloqueada ou perde permanentemente a ligação à Internet, é capaz de recuperar, reiniciando sem intervenção humana. Este requisito deve-se à natureza da *testbed* com as placas instaladas num conjunto de veículos, cujo acesso é muito limitado.

A impossibilidade de utilizar o PXE sem uma ligação cablada corresponde a outro problema que surgiu, impedindo desta forma o carregamento de novas imagens de disco nas placas da *testbed*.

Houve ainda a necessidade de estender o suporte do RC ao driver WAVE, utilizado nas placas da *testbed* do DRIVE-IN. Para isto, foi preciso proceder à alteração de parte do código do RC.

4.5.1 Inclusão de Novos Serviços

Nesta subsecção são descritos os serviços que foram adicionados ao sistema de gestão, que não são fornecidos pelo OMF, de forma a estender a sua aplicabilidade na *testbed* em questão, respondendo a alguns desafios apresentados: a atualização dinâmica do endereço IP, a tolerância a falhas nos nós da *testbed* e o envio de imagens de disco para as placas.

4.5.1.1 IP Dinâmico

A solução para o primeiro problema baseou-se na adição automática e dinâmica do novo IP de controlo das placas na base de dados "inventory" sempre que é feita uma nova ligação,

alterando os serviços do OMF para tornar este procedimento possível. Deste modo, tornou-se possível conhecer em qualquer altura o endereço IP atual da interface 3G, não só aos utilizadores da *testbed*, mas também ao próprio sistema de gestão, que consegue dinamicamente encontrar o endereço IP da placa antes de proceder à execução de uma nova experiência, carregamento de uma nova imagem, ou qualquer outro serviço dependente do endereço IP.

Os *shell scripts* que o sistema operativo usa na gestão das interfaces utilizadas pelo PPP, como é o caso da interface do *modem* USB utilizado, estão contidos na pasta "/etc/ppp/". Nesta pasta pode-se encontrar a subpasta "ip-up.d", juntamente com o *script* "ip-up". Este *script* é executado sempre que é atribuído um novo endereço IP a uma interface PPP que, por sua vez, executa todos os *scripts* que se encontram na subpasta "ip-up.d". A solução encontrada para resolver o problema da atualização do endereço IP nas placas, passou por incluir um novo *script* nesta subpasta, "update-ip-omf". Este *script* faz uma chamada a um serviço HTTP que foi inserido no AM do OMF, enviando-lhe o novo endereço IP, o seu HRN e ainda o domínio em que se encontra o nó da *testbed*.

É conveniente salientar que todos os serviços do AM são, na versão 5.4 do OMF, registados como serviços HTTP e XMPP, cujos servidores são criados aquando do arranque do *daemon*, sendo os referidos serviços acessíveis por ambos os protocolos. Estes serviços estão organizados em classes de serviço, que correspondem às configurações copiadas no passo 3 da instalação do OMF.

O novo serviço, inserido na gama de serviços do AM "inventory", na sua subpasta "ogs_inventory", denomina-se de "setControlIP". Na implementação deste serviço foi acrescentada uma função para fazer o *query* à base dados "inventory", inserindo o IP recebido pelo serviço na coluna "control_ip" da tabela "nodes". Desta forma, o novo endereço IP é sempre atualizado quando há um restabelecimento de ligação, depois duma falha da mesma.

4.5.1.2 Tolerância a Falhas

O problema da tolerância a falhas foi resolvido com o uso do *timer Watchdog* e do respetivo *daemon* de controlo, com o mesmo nome. Este *timer* possibilita reiniciar automaticamente a placa sempre que há um bloqueio ou uma falha de ligação a um determinado endereço IP, por exemplo. Foi necessário configurar o *kernel* de Linux, ativando o módulo "AMD Geode CS5535/CS5536", correspondente ao timer presente na placa, e instalar o *daemon* referido, que foi configurado para enviar um *ping* entre cada 45 segundos ao servidor do IT. Se algum destes *pings* não obtiver resposta ou houver uma falha num teste de escrita no

ficheiro "/dev/watchdog" (que indica que a placa bloqueou), o nó é reiniciado, voltando à imagem de disco base na primeira partição.

Utilizando esta funcionalidade foi simples implementar uma maneira de avaliar se os nós estão ou não ativos, utilizando a base de dados "inventory", conjuntamente com os *pings* recebidos dos nós pelo *Watchdog*. Foi inserido no AM um novo serviço HTTP na gama de serviços "inventory", com o nome de "setLastPing", que se auxilia de uma função que faz um *query* à base de dados, inserindo a data e hora do servidor correspondente ao último *ping* recebido de um dos nós. Os parâmetros desde serviço são o endereço IP de onde veio o *ping* e o domínio da *testbed*. A *query* vai selecionar a linha na tabela "nodes" que contém esse endereço IP, e vai chamar a função "NOW" do MySQL, inserindo a data e hora na coluna "LastPing" correspondente. Para utilizar este serviço, foi desenvolvida uma aplicação que escuta os pacotes recebidos, ciclicamente filtrando os pacotes *echo request* do *Internet Control Message Protocol* (ICMP), retirando-lhes o endereço IP de origem. De seguida, faz uma chamada ao serviço HTTP referido com este endereço IP e o domínio como "default".

Com o desenvolvimento destas aplicações, os utilizadores da *testbed* passaram a ter sempre disponível a informação da atividade das placas, com uma discrepância de 45 segundos, facilitando desta forma, a gestão da *testbed*.

4.5.1.3 Carregamento de Imagens de Disco

Pelo facto de não ser utilizada uma ligação cablada, a utilização do PXE existente no OMF foi desactivada. Sendo assim, tornou-se necessário encontrar uma resolução para o problema de carregamento de imagens. Portanto, tornou-se indispensável encontrar uma forma de carregar as imagens através da nova interface de controlo do sistema de gestão: a ligação banda larga pela rede celular.

Toda a parte do serviço de PXE fornecido pelo OMF foi omissa, tendo sido então considerada a nova funcionalidade que o serviço de gestão deveria implementar para tornar possível o carregamento de imagens remotamente. Neste sentido, o novo serviço deve poder implementar as seguintes funcionalidades:

- Enviar uma imagem do servidor para a placa através da ligação 3G;
- Colocar a nova imagem numa segunda partição da placa;
- Passar as configurações específicas da placa para a nova imagem;
- Fazer o *reboot* da placa na nova imagem.

Para o efeito foi criada no EC uma nova funcionalidade, sendo esta tratada como uma experiência que é carregada aquando o envio do comando "3gload", da mesma forma que funcionam as funcionalidades nativas do EC. Esta aplicação abre uma ligação SSH, a uma ou mais placas, e envia desta forma comandos sequenciais que completam os passos propostos. Tratando-se de uma implementação em Ruby, foi necessário instalar o Ruby *gem net-ssh* [85]. Para auxiliar este processo, a nova imagem é compactada com *Bzip2* ou *Gzip* e colocada na pasta "/var/lib/omf-images-5.4/", onde o serviço "frisbee" do AM faz a validação da existência da imagem sempre que é enviado o comando pelo EC para o seu carregamento num nó. Desta forma, o novo serviço, sempre que chamado, cria primeiro um *symbolic link* na pasta "/var/www/" apontando para a imagem que se pretende carregar. Desta forma, a imagem fica disponível para *download* a partir da placa usando o servidor HTTP Apache 2. A cada um dos nós ao qual se pretende carregar a imagem é aberta uma ligação SSH, usando *threads* para que se processe paralelamente. São então seguidos os passos sequenciais:

1. É verificado se o nó se encontra na primeira partição da unidade de armazenamento, caso contrário o processo é interrompido;
2. A segunda partição de disco ("/dev/sda2") é montada em "/mnt/" e o seu conteúdo apagado;
3. Qualquer imagem de disco com o mesmo nome presente na unidade de armazenamento da placa é apagada;
4. O comando para o *download* da imagem de disco é executado, usando a aplicação Wget;
5. A imagem é descompactada na segunda partição;
6. O ficheiro da imagem de disco compactado em *Bzip2* ou *Gzip* é removido;
7. Depois de descompactada, são copiadas as configurações específicas da placa para segunda partição (o *hostname* e o PIN do cartão utilizado pelo *modem* USB) e é executado um *shell script* de configuração automática das opções de arranque, que inclui a configuração do endereço IP a ser atribuído à interface *ethernet* e a ambas as interfaces da norma IEEE 802.11p, a de controlo e a de serviço.
8. O comando "grub-reboot 1" é executado, sendo que este configura o Grub para fazer o *boot* na segunda partição após o *reboot* seguinte, voltando à primeira automaticamente num *reboot* posterior;

9. O comando *reboot* é executado.

Após estes passos, as placas envolvidas no processo são reiniciadas na nova imagem de disco. Para servir de apoio ao utilizador da *testbed*, foram desenvolvidos dois serviços no AM, que se juntaram ao conjunto de serviços do "inventory" (gestão da base de dados): um que mostra a progressão do *download* da imagem e outro que mostra a última partição em que o nó fez o *boot*.

O primeiro serviço, denominado "setProgress", faz uma *query* na base de dados "inventory", adicionando a percentagem *download* efetuado da imagem de disco na coluna "imaging_progress" do respetivo nó. Para utilizar este serviço foi necessário alterar o processo do carregamento de imagens anteriormente descrito:

1. Antes dos passos já descritos, é executado o comando Wget, com a opção "–spider". Este comando vai gerar um *log* que contém informação do ficheiro remoto: a imagem de disco, incluindo o seu tamanho em *bytes*.

2. O *download* da imagem é iniciado.

3. O *script* "getCompletion" é executado, que tem a utilidade de calcular a percentagem de *download* já realizado. Para isso utiliza a informação obtida no *log* do primeiro comando e compara-a com o tamanho atual da imagem que está a ser descarregada, calculando a percentagem atual.

4. Enquanto este processo é realizado, vai sendo chamado o serviço "setProgress", com os parâmetros: o domínio da *testbed*, o HRN da placa e um número inteiro que representa a percentagem de progresso do *download*.

5. O *script* "getCompletion" é ativado de 10 em 10 segundos durante o processo descrito, até atingir os 100%.

O outro serviço, "setCurrentImage", faz também uma *query* na base de dados "inventory", adicionando o nome da partição em que foi feito o último *boot*, podendo ser "base", se for a primeira partição, ou "custom", se for a segunda, onde são colocadas as novas imagens de disco. A base de dados é atualizada na coluna "current_image" da tabela "nodes" na linha correspondente ao nó em questão. A utilização deste serviço decorre de forma mais simples: foi adicionada ao *shell script* "update-ip-omf", que é executado aquando a atribuição de um

endereço IP na interface PPP da placa, uma chamada a este serviço com os parâmetros: o domínio da *testbed*, o HRN da placa e o nome da partição: "base" ou "custom".

Na figura 4.6 é possível visualizar as alterações feitas na base de dados "inventory", através da interface do *phpMyAdmin*.

id (universally unique id for nodes)	control_ip	control_mac	cmc_ip	hostname	hrn	LastPing	imaging_progress	current_image
1340	46.50.72.58	00:03:2D:08:1A:17	172.16.0.4	drivein81	omf.my.drivein81	2012-06-13 15:37:42	100%	base
1341	89.214.249.194	00:03:2D:08:1A:17	172.16.0.4	drivein82	omf.my.drivein82	2012-06-13 15:38:05	100%	base
1342	89.214.70.98	00:03:2D:08:1A:17	172.16.0.4	drivein83	omf.my.drivein83	2012-06-13 15:37:51	100%	base
1343	92.250.39.218	00:03:2D:08:1A:17	172.16.0.4	drivein84	omf.my.drivein84	2012-06-13 05:48:08	100%	base
1344	46.50.116.111	00:03:2D:08:1A:17	172.16.0.4	drivein85	omf.my.drivein85	2012-06-13 15:37:54	100%	base

Figura 4.6: Alterações feitas na base de dados "inventory"

4.5.1.4 Outros Serviços

Foram ainda acrescentados mais três funcionalidades que são disponibilizados pelo EC: um serviço para fazer *reboot* aos nós para a primeira partição, outro para fazer *reboot*, mas para a segunda partição ("rebootoCustom"), e um último para acrescentar *software* na partição atual da placa ("changeSoftware"). O primeiro e o segundo serviços processam-se de forma idêntica: tal como o serviço de envio de imagens de disco, abrem uma ligação SSH para a placa e enviam os comandos. O primeiro envia apenas o comando de *reboot*, fazendo o próximo *boot* na primeira partição por defeito; o segundo envia, antes do *reboot*, o comando "grub-reboot 1", fazendo assim o próximo *boot* na segunda partição. O terceiro serviço também funciona de forma semelhante: é ativada a ligação SSH e, de seguida, é enviado um ficheiro compactado, contendo a árvore de pastas e os ficheiros que serão instalados na placa ou placas, funcionando da mesma forma que o descrito no "3gload", com a exceção da sequência de progresso do *download* e *reboot*.

Para que o EC possa reconhecer os novos serviços, foi necessário inserir os novos comandos no seu executável: "3gload", "toCustom" e "changeSoftware". Estes foram acompanhados das definições de experiência, que são chamados por estes comandos e chamam, por sua vez, os serviços do AM respetivos. O comando de carregamento de uma imagem é apresentado de seguida:

```
omf 3gload -t omf.my.drivein1,omf.my.drivein2
-i rootfs.tar.bz2
```

em que o carregamento é feito para o nós com os HRN "omf.my.drivein1" e "omf.my.drivein2", com a imagem "rootfs.tar.bz2".

4.5.2 Adaptação ao Driver WAVE

A implementação do protocolo IEEE 802.11p foi realizada no âmbito do projecto DRIVE-IN, pelo grupo de trabalho do IT em Aveiro. Esta impkementação usa o GPS para fazer uma sincronização de canais entre os diferentes nós. Desta forma é fornecido acesso a dois tipos diferentes de canais *wireless*, de controlo e de serviço, como definido no protocolo. Estes canais são acedidos de forma transparente para o utilizador final: é utilizada apenas uma placa *wireless*, que se divide em duas interfaces de rede ("wlan0" e "wlan1"), correspondendo uma à de controlo e a outra à de serviço. O WAVE define sete canais, sendo um para controlo (envio de mensagens críticas e para o anúncio de serviço, usando *WAVE Service Advertisement* (WSA)), e seis para serviço (principalmente para *infotainment*) [75].

Como o OMF não suporta a norma IEEE 802.11p foi necessário alterar o suporte nativo do driver Atheros, dado pelo OMF. Os ficheiros que correspondem às configurações utilizadas pelo RC para configurar as interfaces para as experiências foram alterados, de forma a fornecer este suporte. Os serviços disponibilizados nas placas utilizando a norma IEEE 802.11p são configurados com o auxílio da aplicação *uwme*. Esta aplicação foi desenvolvida no grupo de trabalho do DRIVE-IN e pode ser utilizada para configurar serviços, como o de *user* ou de *provider* de aplicações na VANET.

Quando uma experiência é escrita em OEDL e executada no EC, as linhas de configuração da interface são lidas e enviadas como *string* do EC para o RC contendo, separados por pontos, o nome da interface e a propriedade a ser configurada. Por exemplo:

```
node.net.e0.ip = "192.168.0.2"
```

configura o endereço IP, na interface "eth0" do nó respetivo com o endereço "192.168.0.2". No caso da aplicação *uwme*, as configurações são enviadas num único comando, por exemplo:

```
uwme startUser psid 82-99 req_type accessOnMatch priority 11
wsa_type unsecured channel 172 psc "Accident Alert"
```

configura um *user* na interface IEEE 802.11p. Nesta linha de configuração o "psid" corresponde ao identificador do serviço (um *user* subscreve-se a um *provider* com o mesmo

"psid"); o "req_type" é o tipo de comportamento do *scheduler*, neste caso a "associação" respeita as configurações definidas com a utilização de "accessOnMatch"; a "priority" define o nível de prioridade deste *user*, sendo que um nível mais elevado terá maior prioridade (por exemplo, uma aplicação de segurança deve ser configurada com uma prioridade mais elevada); o "wsa_type" configura o tipo de WSA a ser utilizado para o anúncio dos serviços, "secured" ou "unsecured"; o "channel" define o canal *wireless* em que este serviço é configurado; por fim, o "psc" é uma descrição textual do serviço configurado. Noutro caso:

```
uwme startPS action add psid 03-88 channel 180 access 2 repeat_rate
30 ip_service priority 7 wsa_type unsecure psc "Video Conferencing"
```

configura um serviço de *provider*. Na linha de configuração anterior o comando "action" indica qual operação vai ser realizada, neste caso é adicionado um novo *provider* com o "add"; o "psid" é o mesmo que no caso do *user*, bem como o "channel"; o "access" indica que tipo de acesso deve ser feito aos canais de serviço e controlo (contínuo ou alternado) e neste caso é escolhido o alternado; a "repeat_rate" define o número de envios de WSA por cada 5 segundos, neste caso são enviados 30 por cada 5 segundos; o "ip_service" configura este serviço como um serviço sobre IP; a "priority" tem a mesma função que no caso do *user*, assim como o "wsa_type" e o "psc".

No entanto, do ponto de vista do utilizador, será mais simples enviar as configurações em separado, como no primeiro caso.

A solução encontrada passou pelo uso de um *array* bidimensional. Cada vez que é recebida uma configuração pelo RC, este guarda o seu valor na posição respetiva do *array* para essa configuração e, quando é recebido o comando "startUser" ou "startPS", ou o correspondente a outra configuração, toda a linha do comando é construída e é executado. Este processo permite enviar múltiplas configurações para o mesmo nó, pois cada configuração de propriedade fica guardada numa posição diferente no *array*, correspondente a uma linha que vai ser incluída num comando. As configurações diferentes de cada linha são indicadas pelo utilizador em linguagem OEDL da seguinte forma:

```
node.net.w0.chanPSZ0 = "180"
```

configura o canal "180" no serviço. Na linha de configuração apresentada em "chanPSZ0", a terminação "Z0" indica a qual comando do *uwme* a que esta propriedade corresponde, sendo feito do lado do RC um *split* na letra Z maiúscula, lendo de seguida o número inteiro que se segue, para escolher a posição no *array* a que corresponde. Por exemplo, uma segunda confi-

guração indêntica à apresentada, para um outro comando usando o *uwme*, deveria terminar em "Z1" em vez de "Z0". A última linha introduzida deve ser:

```
node.net.w0.wave_cmd = "startPSZ0"
```

ou o equivalente para o caso de um *user*, ou outra configuração, que indica que o comando do *uwme* deve ser executado.

Atualmente, o uso de "w0" não implica que se esteja a configurar a interface "wlan0", como seria normal. As interfaces específicas "wlan0" e "wlan1" não são utilizadas na configuração destes serviços, estando incluídas na *string* apenas por fazerem parte da linguagem OEDL, sendo utilizadas unicamente para indicar que estamos a configurar a interface *wireless* IEEE 802.11p. Os endereços IP nestas interfaces são estáticos e configurados aquando o arranque, com uma terminação que corresponde ao número atribuido ao nó da *testbed*.

4.6 Método de Recolha de Resultados

As experiências que devem correr na *testbed* são descritas pelos seus utilizador na linguagem OEDL. Nesta linguagem específica do OMF, baseada em Ruby, são descritos os grupos de nós utilizados, as aplicações que vão ser executadas e as suas propriedades, as configurações das interfaces utilizadas e a sequência de passos que a experiência executa, como o arranque e a paragem de aplicações, alteração das suas propriedades dinamicamente ou os tempos de espera entre eventos. Nas propriedades das aplicações são definidas as medições a ser retiradas das experiências.

As medições são retiradas das aplicações com o uso da biblioteca do OML. O OML é munido de um servidor, que corre a par do AM e EC na máquina de servidor do IT, e um cliente, que corre nos nós da *testbed*. Algumas aplicações já implementadas com o cliente OML são disponibilizadas pelo OMF, como o OTG2 e o OTR2, um gerador e um recetor de tráfego, respetivamente, ou uma versão do Iperf.

O cliente OML recolhe as medições definidas na aplicação, enviando-as de seguida ao servidor OML que, por sua vez, guarda estas medições numa base de dados sqlite [70], exclusiva para a experiência em questão.

Não utilizando as aplicações fornecidas, existem duas maneiras de utilizar o OML para recolher medições da aplicação:

- Se a aplicação for escrita em C/C++ e de código aberto, pode ser incluída a biblioteca do OML no próprio código;

- Pode ser feito um *wrapper* para uma aplicação, usando um módulo de Ruby fornecido, o "oml4r", que recolhe o *output* desta aplicação e envia-o ao servidor OML (figura 4.7).

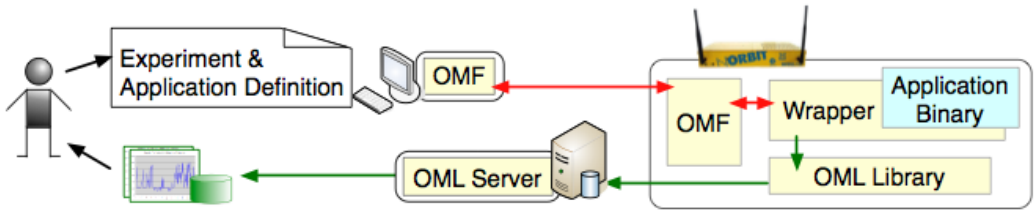


Figura 4.7: Utilização de um *wrapper* com o OML [50]

Os dados resultantes das medições das experiências são armazenados na base de dados em tempo real, podendo ser definido o tempo, em segundos, entre cada envio para o servidor na definição da experiência, ou então, a quantidade de medições a serem recolhidas antes de serem enviadas.

No primeiro caso referido, a alteração do código fonte de uma aplicação escrita em C/C++, dever-se-á proceder à realização dos seguintes passos:

- Incluir a biblioca "oml2/omlc.h" no código;
- Chamar a função "omlc_init()", passando os argumentos de entrada da aplicação, dos quais serão lidos os que começarem por "–oml", antes de se fazer qualquer operação com o próprio código da aplicação;
- Chamar a função "omlc_start()", para arrancar o processo de recolha de resultados, a partir daqui os dados começam a ser recolhidos pelo OML;
- Chamar a função "omlc_inject()" sempre que se querem guardar dados relativos a medições feitas pela experiência;
- Chamar a função "omlc_close()" assim que o programa termina.

Os dados a serem recolhidos num *Measurement Point* (MP) devem ser definidos num *array* de estruturas "OmlMpDef", sendo o primeiro membro desta estrutua o nome da medição recolhida, como *string*, e o segundo o tipo de elemento da mesma, como um *double* ("OML_DOUBLE_VALUE") ou uma *string* ("OML_STRING_VALUE"), por exemplo. Após este processo, deve ser definido o MP propriamente dito, usando a função "omlc_add_mp()",

passando-lhe o seu nome, como *string*, e o *array* previamente definido. Depois deste processo, sempre que se recolherem dados, estes devem ser guardados num *array* de valores do tipo "OmlValueU", usando os *macros* fornecidos para armazenar os valores: "omlc_set_double()", "omlc_set_string()", etc. Por fim, depois de se ter os valores das medições guardados no *array* referido, a função "omlc_inject()" é chamada, passando-lhe o MP onde estes dados deverão ser injetados e o *array* do tipo "OmlValueU" com os valores.

No segundo caso mencionado, é utilizado um *wrapper*, escrito em Ruby, para a aplicação que vai capturar o seu *output* fazer o *parsing* do mesmo, retirando os valores que se pretendem registar, e enviando-os para o servidor OML no final. Para implementar este *wrapper*, primeiro é necessário incluir a biblioteca fornecida, a "oml4r", no código Ruby. A partir daqui, deve ser definido o *schema* do MP que se pretende criar, sendo que este, como no caso anterior, contém a descrição dos dados que se pretendem retirar da aplicação, e nesta descrição são definidos os nomes das variáveis e o seu tipo de dados. De seguida, deve ser definida uma classe com um nome arbitrário, sendo que esta classe deve conter os métodos "initialize", "process_output" e "start".

Na primeira função é feito o *parsing* dos argumentos de entrada no *wrapper* que são guardados em variáveis. É também feita a inicialização da biblioteca "oml4r", permitindo que as medições futuras sejam encaminhadas pela mesma, e é definido o nome da aplicação a que este *wrapper* se destina, que também será o prefixo do nome da base de dados criada pelo servidor OML.

O segundo método, "process_output", tem a utilidade de fazer o *parsing* de cada linha de *output* da aplicação à qual o *wrapper* se destina. Obtendo-se os valores que se desejar, estes deverão ser de seguida injetados no MP criado previamente, usando a função "inject" da biblioteca "oml4r".

Por fim, a função "start" deve, num ciclo infinito, arrancar a aplicação pretendida para este *wrapper*, com os argumentos de entrada previamente guardados. De seguida, irá chamar a função "process_output", usando o *output* da aplicação que se acabou de correr na entrada e voltar ao início do ciclo.

Finalmente, é ainda necessário escrever uma função fora da classe anteriormente descrita. Esta função irá inicializar a classe com os argumentos de entrada do *wrapper* e chamar a função "start".

No final deste processo é ainda indispensável proceder à elaboração de um *Application Definition* (AD) para este *wrapper* ou aplicação no servidor, para que as instruções introdu-

zidas em OEDL, na descrição da experiência, possam ser interpretadas pelo EC. Este AD deve conter todas as propriedades da aplicação e todas as medições que se podem retirar da experiência. Deve ser primeiramente escrita uma declaração que contém o URI e o nome para a aplicação, sendo o URI o local onde este AD reside e o nome a identificação do mesmo. De seguida deve ser dado o caminho absoluto para a aplicação no nó da *testbed* e, opcionalmente, a especificação do método para a instalação desta aplicação, que pode ser um ficheiro *tar* local ou remoto, ou um pacote *deb* ou *rpm*. Pode ser também dada uma descrição textual da aplicação.

Na próxima fase, devem ser definidas as propriedades da aplicação, sendo estes os parâmetros de entrada para ela, que podem ser definidos na definição de um ED. Estas propriedades devem conter o nome do parâmetro, uma descrição, os valores por defeito (podendo ser nulos), o tipo de dados dos mesmos e um *boolean* que define se o parâmetro pode ser alterado durante a execução da aplicação ou não. Por fim, devem ser definidos os MP da aplicação, juntamente com as medições que se podem retirar dos mesmos, sendo que estas definições devem conter o nome da medição e o seu tipo de dados.

Por fim, no ED, pode-se definir o uso da aplicação previamente instrumentada, usando o AD elaborado, e fornecendo as propriedades a serem configuradas e os MP de onde se pretendem retirar os resultados, assim como a frequência de envio dos mesmos [50] [71].

4.7 Imagem das Placas

Devido à necessidade de se ter uma imagem pequena o suficiente para ser enviada pela rede 3G para as placas da *testbed*, tornou-se imprescindível encontrar uma distribuião pequena, ou gerar uma imagem em disco com dimensões reduzidas.

Embora distribuições como Damn Small Linux [86] ou TinyCore [87] ofereçam um tamanho reduzido, estas não foram consideradas como ideais, pois não disponibilizam a mesma liberdade caso se crie uma imagem de disco de raiz, sendo esta a opção escolhida.

4.7.1 Método e Critérios de Criação

Para auxiliar a criação da imagem de disco foi utilizada a ferramenta *Buildroot* [88]. Esta ferramenta é capaz de gerar um *toolchain* para *cross-compilation*, um *root filesystem*, uma imagem de *kernel* e uma imagem de *bootloader*. O *Buildroot* suporta várias arquiteturas de CPU e é útil para quem trabalhe com pequenos sistemas ou sistemas embebidos.

Os motivos que levaram à escolha do *Buildroot* foram:

- A liberdade que oferece para a geração da imagem;
- A facilidade de configuração, potenciando maior rapidez no desenvolvimento de trabalho;
- A quantidade de pacotes de *software* suportados nativamente, incluindo muitos essenciais para a imagem das placas;
- O suporte de vários tipos de sistemas de ficheiros para a imagem gerada;
- O suporte para *uClibc* [89];
- A sua estrutura simples que facilita a sua extensão, como inclusão de novas aplicações, de forma rápida [88].

Da mesma forma que se configura o *kernel*, pode-se usar o "menuconfig" no *Buildroot*, e, desta forma, selecionar todas as configurações desejadas para a imagem gerada. Na compilação foi utilizada a biblioteca de C *uClibc*, versão 0.9.32, que é uma biblioteca C compacta para sistemas Linux embebidos. Esta biblioteca é muito mais pequena que a biblioteca C do GNU (*glibc*) e suporta quase todas as mesmas aplicações, bastando uma recompilação do código fonte. O *uClibc* suporta também bibliotecas partilhadas, *threading* e uma elevada variedade de arquiteturas de CPU [89]. Foi instalado também o *Busybox* [90] (versão 1.19), que combina versões pequenas de *utilities* comuns no UNIX, como as *fileutils* ou as *shellutils*, num executável pequeno e único. Embora estas opções sejam mais limitadas que nas versões GNU, estas oferecem uma funcionalidade e um comportamento muito semelhante ao que se encontra nas *utilities* correspondentes do GNU [90].

O *kernel* de Linux, instalado na imagem, corresponde à versão 3.3.7 modificada para suportar a versão do *driver* "ath5k", por sua vez alterada para suportar WAVE [75], sendo que este *driver* também foi instalado na imagem. Foi necessário proceder a algumas configurações extra para o suporte do *modem* USB e ao *Watchdog*, já referidas nas secções 4.4 e 4.5.1, respetivamente.

O *uClibc*, embora reduza o tamanho da imagem, impede que aplicações compiladas usando outra biblioteca C corram na imagem, mesmo usando um CPU com a mesma arquitetura. É portanto necessário compilar as aplicações com esta biblioteca, havendo três formas de o fazer:

- Compilar a aplicação junto com todo o processo de configuração e compilação automática do *Buildroot*, obtendo a aplicação dos respetivos repositórios, ou então colocando-a manualmente na respetiva pasta;
- Compilar a aplicação numa placa que esteja a usar já uma imagem gerada no *Buildroot*;
- Compilar a aplicação noutra máquina, usando o método de *cross-compile* e a versão do GCC e/ou G++ compilada para *uClibc*, bem como todas as ferramentas utilizadas na compilação que dependam da biblioteca de C, como o LD ou o AR.

É de notar que, no primeiro caso, é essencial proceder a uma elaboração dos Makefile e alteração dos ficheiros de configuração do *Buildroot*. Embora não seja uma tarefa muito complexa, requer mais tempo que nos outros dois casos, sendo esses então os preferidos sempre que foi preciso compilar uma aplicação para a imagem que não vinha incluída no *Buildroot*.

4.7.2 Aplicações Instaladas

Foram instaladas algumas aplicações suportadas de raiz pelo *Buildroot*, sendo que as mais relevantes são:

- *Tar*, *Bzip2* e *Gzip*: para a descompactação das imagens recebidas nas placas;
- *Ruby*: linguagem de interpretação utilizada pelo OMF;
- *pppd*: aplicação *daemon* utilizada para estabelecer as ligações *dial-up* usando o protocolo PPP;
- *Dropbear*: servidor e cliente SSH;
- *Wget*: aplicação utilizada para fazer o *download* das imagens de disco;
- *Grub*: o *bootloader* utilizado;
- *Net-snmp*: servidor SNMP, para funcionalidades futuras da *testbed*;
- *udev*: versão 164, utilizado na secção 4.8.

Algumas aplicações tiveram que ser compiladas manualmente, pois o *Buildroot* não lhes oferece suporte:

- A aplicação *iw*, de configuração das interfaces *wireless*, modificada para o suporte ao driver WAVE;

- As aplicações do OML, versão 2.6.1, OTG, OTR, *GPS-Logger* e *Iperf*;
- As bibliotecas do OML também foram compiladas para a placa;
- O RC do OMF foi instalado na placa;
- O *usb_modeswitch*, embora suportado pelo *Buildroot*, dispõe de uma versão antiga e, por isso, foi compilada a versão 1.2.3;
- O *daemon Watchdog* foi compilado para a placa, cuja versão fornecida pelo *Buildroot* não oferece as mesmas capacidades.

Na imagem foram colocados os *scripts* de arranque, que fazem o carregamento dos módulos do *kernel*, a ligação à rede 3G e o arranque dos *daemons* utilizados, bem como as configurações de IP nas interfaces *ethernet* e IEEE 802.11p. O *script* que inicia a ligação à rede móvel processa-se da seguinte forma:

- É executado o *usb_modeswitch*, usando o ficheiro de configuração como parâmetro;
- Espera que os ficheiros do dispositivo USB sejam criados na pasta "/dev/";
- É efetuada a ligação *dial-up*.

No final, a imagem gerada ficou com um tamanho de cerca de 50 MB, compactada em *Bzip2*, possibilitando o seu envio através da rede 3G e não ocupando demasiado espaço nos cartões *CompactFlash* de 4 GB, utilizados nas placas da *testbed*.

4.8 Acesso à Internet nos Táxis

Para que os clientes e os motoristas da frota de táxis possam ter acesso aos serviços de Internet, através de qualquer dispositivo munido de uma interface *wireless*, foi feita uma partilha da ligação 3G através de uma interface Wi-Fi na placa. Para o efeito, foi utilizada uma placa Wi-Fi com a norma IEEE 802.11n USB. Para tal foi activado o driver da *Atheros* "ath9k_htc" nas placas. Também foi necessária a instalação do *udev*, que não estava presente nas imagens de disco para poupar algum espaço, pois é utilizado para localizar e carregar o *firmware* desta placa *wireless*.

Para a implementação foram utilizadas três aplicações: o *Hostapd* [91], o *daemon* de DHCP DHCPD e o *IPTables*. O *Hostapd* é um *daemon* utilizado para criar um *Access-Point* (AP)

e servidores de autenticação, sendo configurado na interface da placa Wi-Fi IEEE 802.11n. O DHCPD, o *daemon* utilizado para criar um servidor DHCP, foi configurado para atribuir o endereço IP aos clientes que se ligam ao AP. Por fim, foi necessário configurar o *IPTables* para usar NAT, para fazer a tradução entre endereços locais da rede Wi-Fi e a Internet.

A figura 4.8 exemplica esta funcionalidade.

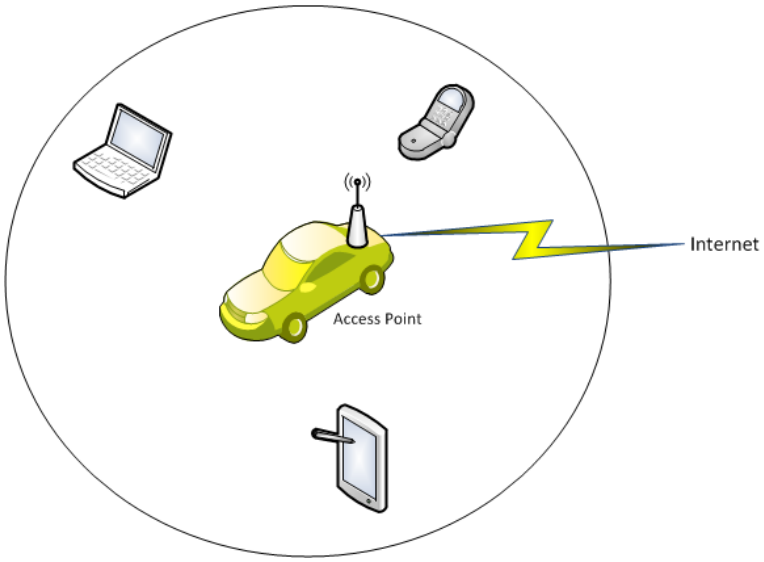


Figura 4.8: Partilha de Internet nos táxis

De seguida, foi necessário encontrar uma solução para a gestão dos utilizadores deste serviço. Para tal, foi utilizado o OML, para enviar a informação dos acessos à rede para o servidor. Conforme foi descrito, é possivel instrumentar aplicações para funcionarem como clientes OML, usando um *wrapper* para o ler e enviar o *output* do programa, ou incluindo a biblioteca do OML no código C ou C++ e usando as suas funções. Para esta implementação foi tomada a segunda opção. Foi portanto feita uma aplicação simples em C++, que lê a informação dos acessos à rede e a envia para o servidor localizado no IT. Sendo esta aplicação implementada como se se tratasse de uma experiência normal para o OMF, a informação fica guardada numa base de dados em *sqlite*, única e exclusiva para o nó da *testbed* em questão.

As informações das ligações dos utilizadores ao AP são obtidas a partir dos *logs* gerados pelo *Hostapd* e pelo DHCPD, sendo que estes foram ativados nas suas configurações e são escritos no ficheiro "messages", localizado em "/var/log/". A referida aplicação faz uma inicialização do MP de onde se vão obter os dados, sendo que este contém as variáveis para guardar o *timestamp* do evento (conforme apresentado no ficheiro de *log*), o endereço MAC do utilizador e, quando aplicável, o *hostname* da sua máquina e o endereço IP atribuído pelo

DHCP. Também é guardado o tipo de evento referente à informação, que pode ser um evento do DHCPD (atribuição de IP), ou do *Hostapd* (associação e desassociação do utilizador). O ficheiro de *log* é lido continuamente entre cada 10 segundos e, sempre que surge uma nova linha, esta é analisada. Se a linha lida contiver informação relevante, os dados são guardados nas variáveis e injetados para o OML, que procedará ao seu envio para o servidor OML. A aplicação é colocada a correr no arranque do nó, com os parâmentros requeridos para o seu funcionamento: o "oml-id", o "oml-exp-id", cujo nome corresponde ao nome da base de dados gerada no servidor, e o "oml-server", que corresponde ao URI do servidor e porto utilizado pelo OML. Desta forma é sempre possível ver a informação de *log* dos acessos feitos nas diferentes placas que compõem a *testbed*, acedendo à respetiva base de dados (figura 4.9).

event	timestamp	ip	client	protocol	mac_address
CONNECTED	Jun 26 15:10:15	null	null	802.11n	4c:0f:6e:90:0d:9b
DHCP	Jun 26 15:10:18	30.0.0.102	joao-ubuntu-laptop	802.11n	4c:0f:6e:90:0d:9b
DISCONNECTED	Jun 26 15:13:14	null	null	802.11n	4c:0f:6e:90:0d:9b

Figura 4.9: Secção da base de dados com as informações dos acessos à rede *wireless* num táxi

4.9 Conclusão

Ao longo deste capítulo foi descrita a forma como o sistema de gestão OMF foi implementado para a *testbed* do DRIVE-IN. Primeiro, foi descrita a arquitetura do sistema a integrar na *testbed*, de forma a se conhecer melhor as necessidades para a implementação, assim como o material que foi utilizado para integrar nos nós da *testbed* desta rede veicular. Foi assim apresentada a solução para a conetividade dos nós à rede de controlo do OMF com o uso da rede celular. De seguida, foi explicada a instalação do OMF nas placas que compõem os nós e a sua configuração inicial. Foi então exposta a maneira como se instalaram os *modems* USB da rede 3G nos nós, que à partida seriam incompatíveis com o sistem operativo utilizado, com o uso das aplicações *usb_modeswitch* e *pppd*, para fazerem o reconhecimento do *modem* e a ligação *dial-up* à rede, respetivamente. Num passo seguinte, procedeu-se à extensão de funcionalidades do sistema de gestão, para garantir o seu bom funcionamento na rede veicular do DRIVE-IN. Na referida secção foi dada resposta a alguns desafios que surgiram com a instalação do OMF nesta rede:

- O problema do endereço de IP dinâmico obtido a partir da rede 3G foi resolvido com a criação de um serviço que é chamado sempre que uma nova ligação é feita num nó, que

insere o seu IP na base de dados do OMF;

- O problema das falhas de ligação e bloqueio das placas foi colmatado com o uso do *Watchdog*, que reinicia os nós sempre que existe uma falha de conetividade com o servidor remoto ou quando há um bloqueio que impede o funcionamento do mesmo;

- O problema do carregamento de imagens sem o uso de PXE foi resolvido com o uso de uma imagem de disco de dimensões reduzidas, que é enviada para as placas e colocada numa segunda partição de disco, de forma a evitar falhas na partição principal, usando SSH para a comunicação com os nós;

- O problema da incompatibilidade com o *driver* da interface da norma IEEE 802.11p foi resolvido com a alteração do código fonte do RC, de forma a inserir as configurações para esta interface durante a realização de experiências.

Este capítulo apresentou também a forma como se podem recolher os dados de experiências, ativadas com o auxílio do sistema de gestão, numa base de dados localizada no servidor. Após esta explicação foi também apresentada a forma como foram geradas as imagens de disco que são utilizadas nas placas que compõem a *testbed*, sendo estas imagens de dimensões reduzidas geradas com a aplicação *Buildroot* e usando a biblioteca de C *uClibc*, instalando apenas as aplicações necessárias na referida imagem. Finalmente, foi apresentada a forma como a ligação à Internet foi partilhada nos táxis da RadiTáxis, bem como a forma como os dados relativos aos acessos à rede disponibilizada são registados na base de dados do servidor.

Após a realização desta implementação, pode-se concluir que a implementação de um sistema de gestão, como o OMF, numa rede veicular como a do DRIVE-IN é exequível, embora com um conjunto significativo de extensões.

Capítulo 5

Resultados

Neste capítulo são apresentados os resultados, obtidos em laboratório e em ambientes reais, com diferentes experiências usando o sistema de gestão da *testbed*.

5.1 Resultados Obtidos em Laboratório

5.1.1 Envio de Imagem

Nesta subsecção são apresentados os resultados de desempenho do sistema no envio de imagens para as placas. O envio de uma imagem de disco foi efetuado e analisado para situações com o envio para várias placas em simultâneo, com o comando (exemplo):

```
omf 3gload -t omf.my.drivein81, omf.my.drivein82,
omf.my.drivein83, omf.my.drivein84 -i rootfs.tar.gz
```

Foram analisados os tempos obtidos desde o início do envio desta imagem até ao fim da operação, antes do *reboot*, usando a o comando "time" do Linux. Foram testados dois métodos de compactação da imagem: utilizado *Gzip* e *Bzip2*. A imagem tinha um tamanho de 60 MB no caso da compressão com *Gzip* e 53 MB no caso com *Bzip2*.

Nas tabelas pode ser observado, do lado dos nós, o tempo do *download* e extração do arquivo e, do lado do servidor, o tempo que o comando "3gload" demorou a ser executado, i.e., o tempo de toda a operação, desde o envio da imagem até ao envio do comando de *reboot*, o que corresponde à coluna "Execução" nas tabelas apresentadas nesta subsecção.

Para auxiliar estes testes foi criada uma página que apresenta o HRN dos nós que se encontram ativos, a imagem atual e a percentagem de *download* realizado, quando aplicável, em cada uma delas. Na figura 5.1 apresenta-se o envio de duas imagens de disco em simultâneo.

Na figura 5.2 é apresentado o estado dos nós depois de um envio com sucesso, já depois de terem feito o *boot* na segunda partição, na mesma interface. Como demonstra a figura 5.1, a percetagem de *download* obtida da base de dados do OMF é apresentada na interface durante o processo de envio de imagens de disco. Na figura 5.2 observa-se que os dois nós que estavam a realizar o *download* na figura anterior realizaram o *boot* na partição *custom*.

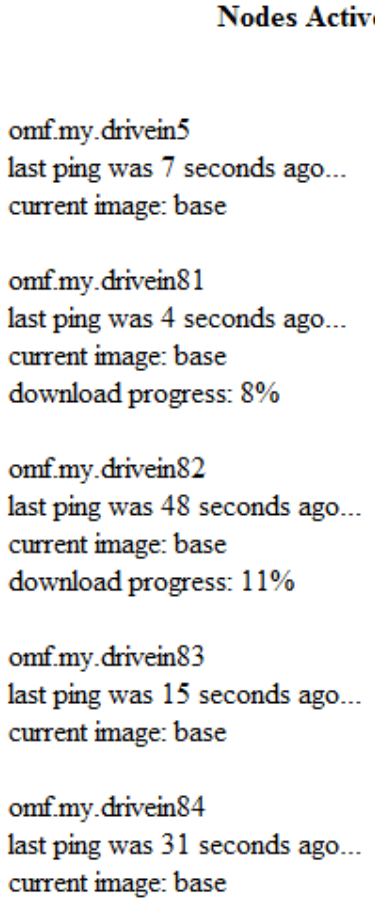


Figura 5.1: Demontração do envio de imagens de disco para os nós da *testbed*

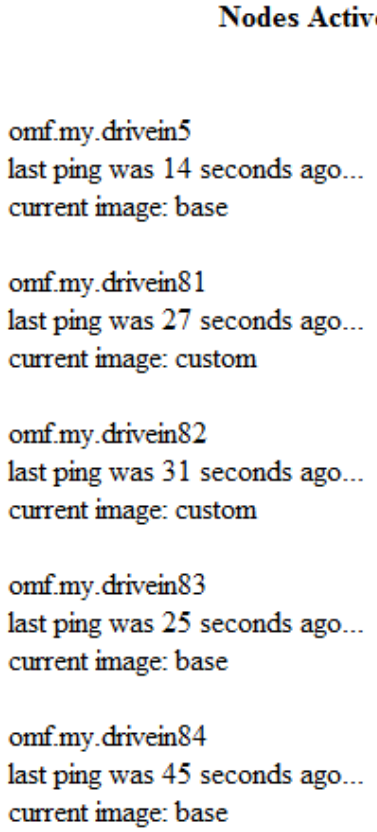


Figura 5.2: Estado dos nós da *testbed* depois de um envio de imagem

Em primeiro lugar foram feitos seis testes com o envio de uma imagem de disco para um dos nós. Os resultados podem ser observados nas tabelas 5.1 e 5.3. Os resultados estatísticos relativos às médias, desvios-padrão e intervalos de confiança de 95% observam-se nas tabeas 5.2 e 5.4, para o caso do envio da imagem comprimida com *Bzip2* e *Gzip*, respetivamente. Nas figuras 5.3 e 5.4 são apresentados as representações gráficas dos valores obtidos nestas experiências.

Tabela 5.1: Tempos com o envio em *Bzip2* para um nó

Download **(segundos)**	**Extração (segundos)**	**Execução (segundos)**
240,738	122,753	416,711
265,297	123,421	432,006
293,936	123,166	481,670
186,238	92,536	366,801
179,194	91,685	336,766
248,668	91,490	421,823

Tabela 5.2: Dados estatísticos relativos aos tempos com envio em *Bzip2* para um nó

	Download	**Extração**	**Execução**
Média e desvio-padrão	235,679 ± 44,945	107,509 ± 17,099	409,296 ± 51,053
Intervalo de Confiança 95%	]188,511;282,846[	]89,564;125,453[	]336,766;481,670[

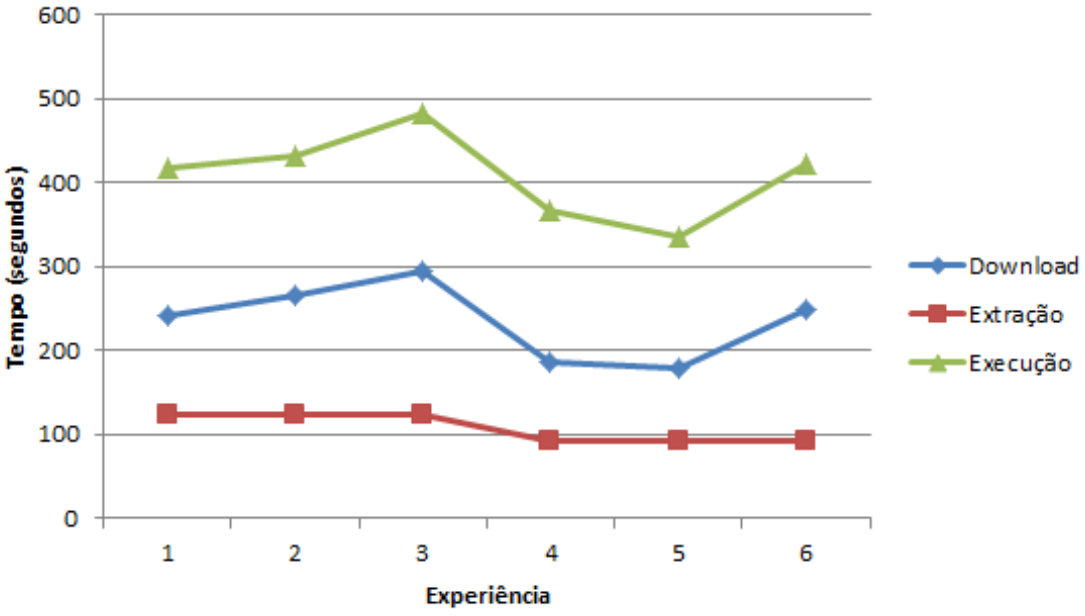


Figura 5.3: Tempos com o envio em *Bzip2* para um nó

Tabela 5.3: Tempos com o envio em *Gzip* para um nó

Download **(segundos)**	**Extração (segundos)**	**Execução (segundos)**
362,444	33,267	461,815
273,55	32,737	361,76
304,145	35,145	401,847
382,001	33,213	466,753
196,746	27,839	281,665
257,76	28,35	331,7

Tabela 5.4: Dados estatísticos relativos aos tempos com envio em *Gzip* para um nó

	Download	**Extração**	**Execução**
Média e desvio-padrão	296,107 ± 68,840	31,759 ± 2,960	384,257 ± 73,381
Intervalo de Confiança 95%	]223,864;368,351[	]28,653;34,864[	]307,248;461,265[

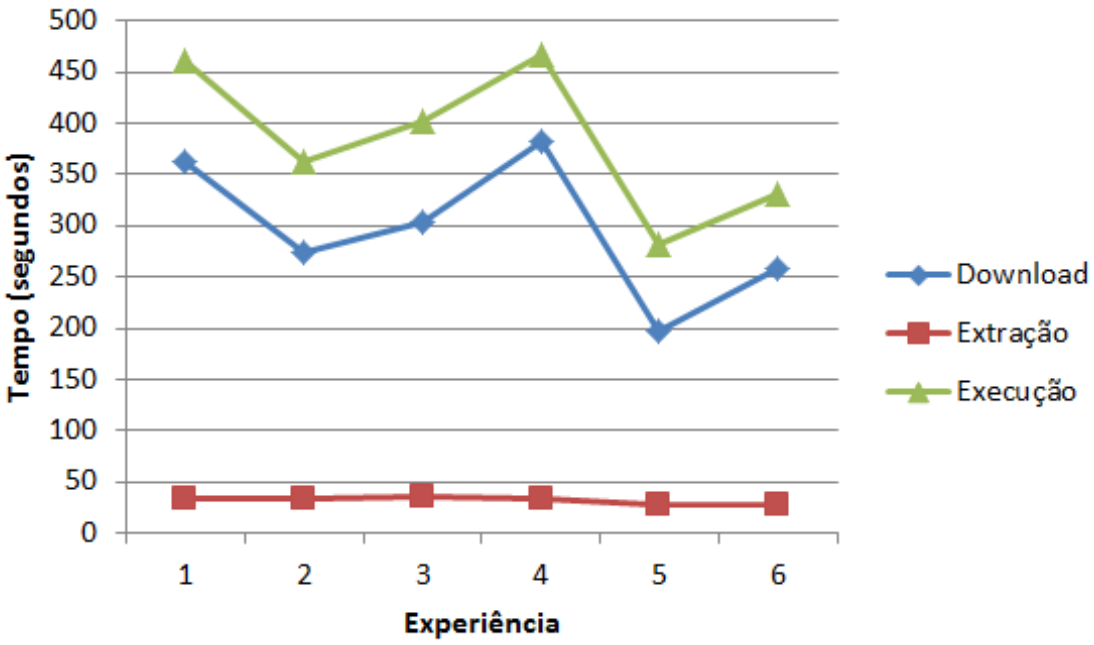


Figura 5.4: Tempos com o envio em *Gzip* para um nó

Pode-se concluir por estes resultados que existe uma grande variação nos tempos de *download*, o que demonstra que a qualidade de serviço da rede celular é muito variável, podendo-se obter resultados muito diferentes. Como por cada envio de imagem acontece um *reboot* no final, cada vez que este envio foi realizado ocorreu uma nova ligação à rede 3G, podendo ficar com características de qualidade de serviço diferentes das existentes na experiência anterior.

Pelos resultados obtidos, a diferença de tempos entre os dois tipos de compressão não é muito significativa. Se no caso do *Gzip* o arquivo é um pouco maior, no caso do *Bzip2* o tempo de descompressão é superior, obtendo-se resultados totais não muito diferentes.

Foi testado seguidamente o envio de uma imagem para dois nós em simultâneo, como pode ser visto nas tabelas 5.5 e 5.6.

Tabela 5.5: Tempos de envio em *Bzip2* para dois nós em simultâneo

HRN	*Download* (segundos)	Extração (segundos)	Execução (segundos)
Om.my.drivein81	362,252	123,053	587, 646
Om.my.drivein82	251,670	121,752	

Tabela 5.6: Tempos de envio em *Gzip* para dois nós em simultâneo

HRN	*Download* (segundos)	Extração (segundos)	Execução (segundos)
om.my.drivein81	255,452	32,948	341,656
om.my.drivein82	202,505	33,381	

É possivel verificar o mesmo tipo de variação da qualidade de serviço que no primeiro caso. Desta vez, obteve-se um tempo melhor com o caso do *Gzip*, não tendo havido qualquer vantagem no uso de *Bzip2*.

Por fim, foi realizado um teste de envio em tudo idêntico mas para quatro nós, como se pode ver nas tabelas 5.7 e 5.8.

Tabela 5.7: Tempos de envio em *Bzip2* para quatro nós em simultâneo

HRN	*Download* (segundos)	Extração (segundos)	Execução (segundos)
om.my.drivein81	956,316	123,573	1127,632
om.my.drivein82	187,472	123,088	
om.my.drivein83	729,372	112,084	
om.my.drivein84	589,968	116,112	

Tabela 5.8: Tempos de envio em *Gzip* para quatro nós em simultâneo

HRN	*Download* (segundos)	Extração (segundos)	Execução (segundos)
om.my.drivein81	1292,944	31,831	1507,350
om.my.drivein82	765,528	31,900	
om.my.drivein83	1055,520	27,938	
om.my.drivein84	1347,527	27,645	

Neste último caso é possível verificar que, embora as experiências tenham corrido sem mais problemas, os tempos de *download* subiram significativamente. Isto deve-se, possivelmente, ao facto do teste ter sido realizado dentro do laboratório, onde os *modem* 3G se encontravam próximos uns dos outros, portanto na mesma célula da rede celular, o que, possivelmente, reduziu a qualidade de serviço. Testes realizados com *downloads* simultâneos não relacionados com o sistema de gestão utilizando a ligação 3G resultaram em velocidades *download* baixas, indicando que é improvável que o problema esteja relacionado com o sistema de gestão. É possível ver que num dos casos o *download* processou-se normalmente (na placa "omf.my.drivein82" do caso com *Bzip2*), embora nos outros três do mesmo teste também tenha ocorrido a mesma situação. Neste caso o teste com *Gzip* foi mais lento, mesmo contando com o tempo de extração do arquivo em *Bzip2*. No gráfico 5.5 observa-se a comparação dos tempos de *download* nos dois casos analisados, no qual se comprova que o tempo de *download* com a imagem compactada em *Bzip2* foi menor, apesar de o resultado da extração do arquivo ser maior, traduzindo-se num tempo de execução mais elevado.

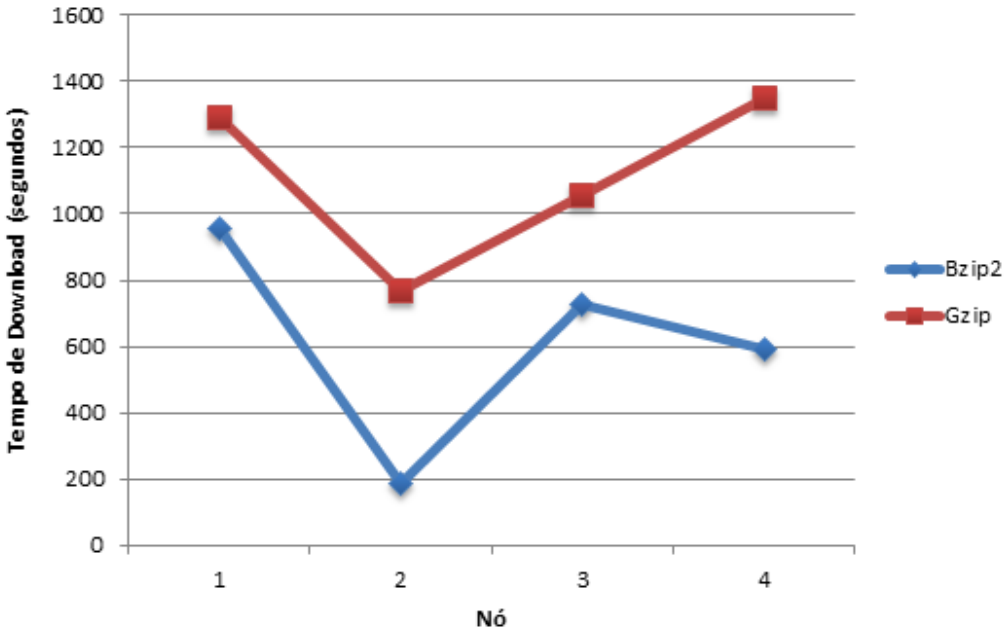


Figura 5.5: Tempos de *download* de quatro nós em simultâneo

5.1.2 Transmissão de Dados

Esta subsecção apresenta a avaliação de desempenho do sistema no processo de transmissão de dados nas experiências escalonadas. As experiências seguintes consistiram no envio de tráfego de um nó para vários nós e de vários nós para um. É de notar que a impossibilidade de utilização do GPS no ambiente de laboratório inviabilizou a sincronização dos canais do WAVE, possibilitando a ocorrência de grandes taxas de perda de pacotes.

As experiências foram escritas em OEDL e foram utilizadas as aplicações OTG2 e OTR2, gerador e recetor de tráfego, para obter os resultados apresentados.

O exemplo de um bloco de código onde é definido o nó recetor, a configuração da aplicação utilizada para a receção do tráfego e a configuração de *user* na interface WAVE pode ser analisado na figura 5.6.

```
defGroup('Receiver', 'omf.my.drivein82') do |node|
  app2="otr2"
  node.addApplication("test:app:otr2") do |app|
    app.setProperty('udp:local_host', '20.0.0.82')
    app.setProperty('udp:local_port', 3000)
    app.measure('udp_in', :samples => 3)
  end
  node.net.w0.psidUserZ0 = "81-82"
  node.net.w0.priorityUserZ0 = "5"
  node.net.w0.wave_cmd = "startUserZ0"
end
```

Figura 5.6: Configuração do nó recetor em OEDL

Em primeiro lugar, foi definido o HRN do recetor, "omf.my.drivein82", de seguida foi configurada a aplicação OTR2 com as propriedades "UDP:local_host" e "UDP:local_port", bem como o MP a ser utilizado para enviar os dados da aplicação, "udp_in", com uma recolha de 3 *samples* de medições antes do seu envio. Por fim, é configurada o *user* na interface WAVE, com o "psid" 81-82 e a "priority" a 5.

A configuração de um dos nós que envia pacotes UDP é a exemplificada na figura 5.7.

```
defGroup('Sender', 'omf.my.drivein81') do |node|
  app1="otg2"
  node.addApplication("test:app:otg2") do |app|
    app.setProperty('udp:local_host', '20.0.0.81')
    app.setProperty('udp:dst_host', '20.0.0.82')
    app.setProperty('udp:dst_port', 3000)
    app.setProperty('cbr:size', 128)
    app.setProperty('cbr:rate', 1024)
    app.measure('udp_out', :samples => 3)
  end
  node.net.w0.chanPSZ0 = "180"
  node.net.w0.rateZ0 = "12"
  node.net.w0.txpowerZ0 = "23"
  node.net.w0.wave_cmd = "insertChanZ0"

  node.net.w0.psidPSZ0 = "81-82"
  node.net.w0.chanPSZ0 = "180"
  node.net.w0.priorityPSZ0 = "7"
  node.net.w0.psc = "experiment2"
  node.net.w0.wave_cmd = "startPSZ0"
end
```

Figura 5.7: Configuração de um nó emissor em OEDL

Nesta figura pode ser visualizada uma configuração semelhante com a anterior, mas agora para o nó "omf.my.drivein81" e a aplicação é a OTG2, que deverá enviar pacotes para o nó anterior. Também se configura a taxa de transmissão e o tamanho dos pacotes, neste caso com 1024 bps e 128 bytes, respetivamente. Neste caso o MP é o "udp_out" e o serviço configurado

na interface WAVE é o de *provider*, sendo também configurado o canal da interface de serviço.

Por fim, pode ser observado na figura 5.8 a sequência em que as aplicações são iniciadas e terminadas para a realização da experiência.

```
onEvent(:ALL_UP_AND_INSTALLED) do |event|
  group('Receiver').startApplications
  wait 10
  group('Sender1').startApplications
  group('Sender2').startApplications
  group('Sender3').startApplications
  wait 60
  group('Sender1').stopApplications
  group('Sender2').stopApplications
  group('Sender3').stopApplications
  group('Receiver').stopApplications
  Experiment.done
end
```

Figura 5.8: Sequência da experiência em OEDL

Deste modo, primeiro é ativado o OTR2 no recetor e após 10 segundos é ativado o OTG2 nos 3 nós que enviam o tráfego. A experiência decorre durante 60 segundos e são terminadas todas as aplicações antes de a experiência terminar.

Com o serviço disponibilizado pelo OMF, "omf-web", é possível aceder aos resultados, compostos pelos gráficos e tabelas com os dados utilizados para os gerar. A informação mais detalhada dos dados recolhidos na experiência pode ser obtida acedendo à base de dados *sqlite*, que é gerada na pasta "/var/lib/oml2/" pelo servidor OML.

Na primeira experiência realizada foram enviados pacotes UDP de 3 placas para uma outra, usando a interface WAVE. Foi utilizada uma taxa de transmissão de 1024 bps e pacotes de 128 bytes, decorrendo numa duração de 60 segundos. Todas as medições obtidas nas experiência são enviadas para o servidor, configurando o "samples" com o valor 1 para o efeito.

Na figura 5.9 é apresentado o gráfico gerado para os nós que enviaram tráfego. Cada linha representa um dos nós, no entanto estas estão sobrepostas, pois os dados são muito próximos. O "oml_seq" é um *sequence number* gerado para os pacotes recebidos e enviados pelo OML, sendo que os mesmos ficam marcados com a ordem de chegada à base de dados do servidor, tornando-se desta forma mais fácil apresentar o gráfico dos resultados. O eixo dos xx representa o *timestamp*, em segundos, medido no servidor, e o dos yy representa o "oml_seq".

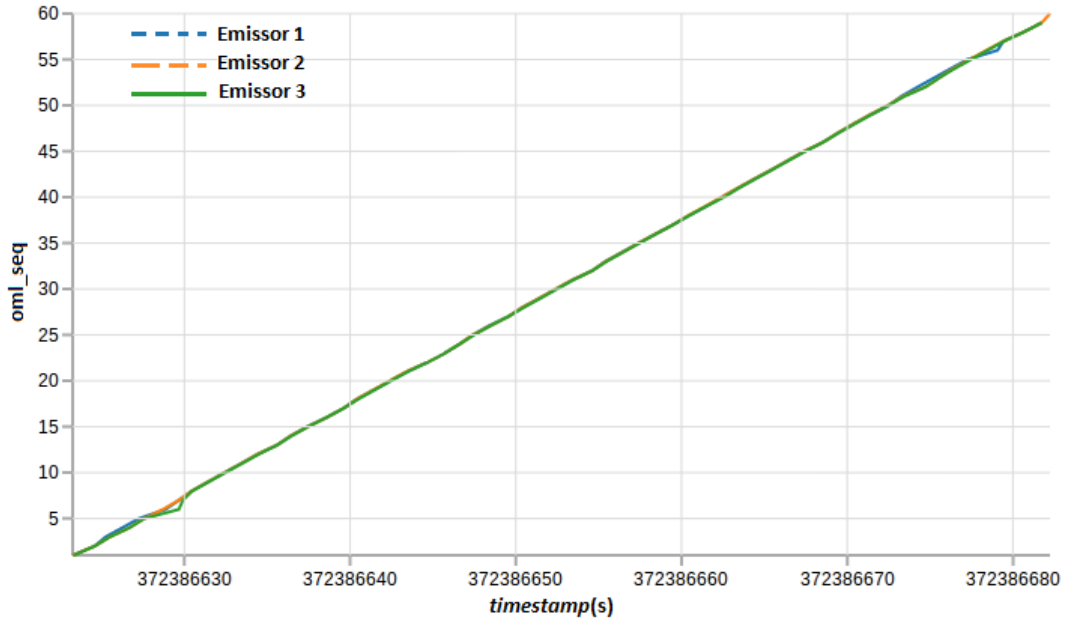


Figura 5.9: Gráfico dos três nós emissores no primeiro caso

Na figura 5.10 pode ser observado o gráfico gerado para o recetor, neste caso cada linha representa o nó de origem do pacote recebido. O eixo dos xx representa o *timestamp* e o dos yy o "oml_seq".

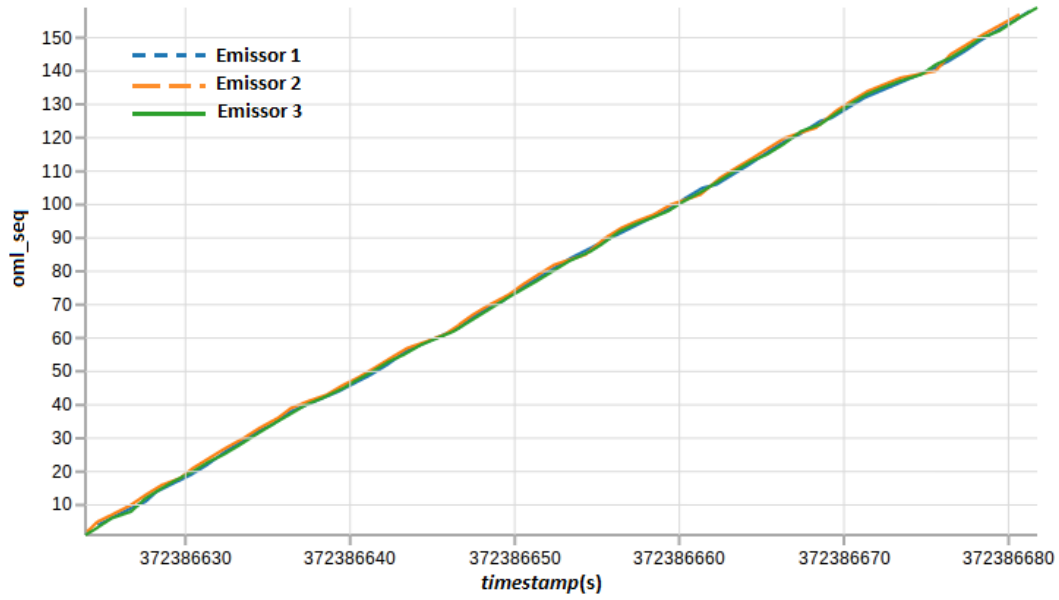


Figura 5.10: Gráfico do nó recetor no primeiro caso

Observando estes gráficos é possível concluir que a maior parte dos pacotes chegou ao seu destino no instante especificado. No entanto, acedendo à base de dados e calculando a taxa de perda de pacotes, observou-se que esta foi de 10,67%.

A mesma experiência foi repetida, agora com uma taxa de transmissão de 2048 bps. Na figura 5.11 apresenta-se o gráfico dos nós que enviaram os pacotes UDP e na figura 5.12 apresentam-se os resultados do recetor, em que cada linha representa um dos nós emissores e um dos nós de origem dos pacotes, respetivamente. Em ambos os casos o eixo dos xx representa o *timestamp* e o dos yy o "oml_seq".

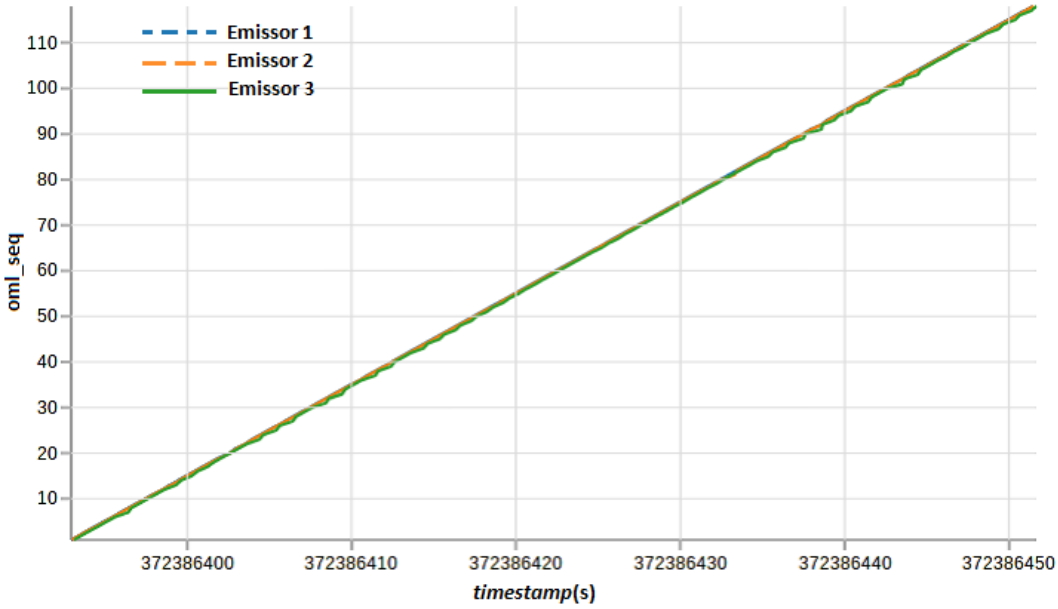


Figura 5.11: Gráfico dos três nós emissores no segundo caso

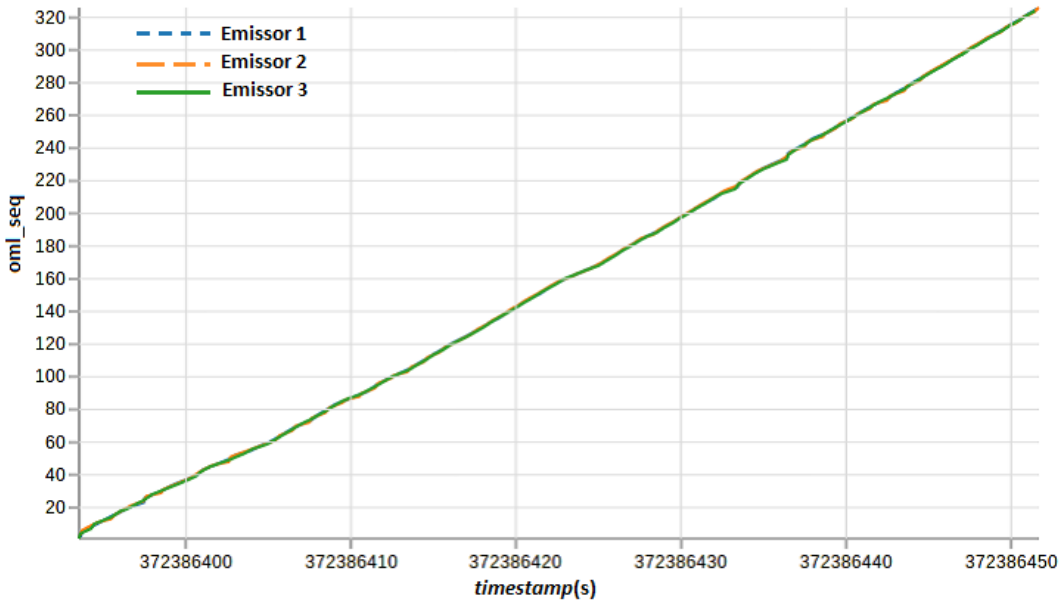


Figura 5.12: Gráfico do nó recetor no segundo caso

Neste caso houve uma taxa de perda de pacotes de 7.91%. Pode-se concluir que nesta situação, onde se utilizou uma *bit rate* mais elevada, houve uma menor percentagem de perda de pacotes que no caso anterior, onde se utilizou uma *bit rate* menor. Nestas experiências não existe uma sincronização dos canais da interface *wireless* entre os diferentes nós, pois não se utilizou o módulo de GPS para o efeito destas experiências realizadas em laboratório. A menor percetagem de perda de pacotes pode estar relacionada com o facto de a *bit rate* ser maior, portanto a probabilidade dos pacotes de dados, quando são transmitidos, acertarem no canal certo é mais elevada. Como se utiliza UDP, não acontece uma retransmissão dos pacotes perdidos, como aconterecia em TCP. Assim, sempre que existe uma falha na sincronização, o pacote perdido nesse instante nunca será recebido no nó recetor.

Repetindo as configurações destas experiências, mas com um nó a enviar pacotes UDP em *broadcast* e três nós a receber tráfego, observaram-se os resultados das figuras 5.13 e 5.14. A taxa de transmissão é de 1024 bps e os pacotes têm um tamanho de 128 bytes. O valor do "samples" é o mesmo que nas experiências anteriores. No gráfico em 5.14 cada linha representa um dos nós, enquanto que no 5.13 apenas existe uma linha que representa o nó emissor e, em ambos os casos, o eixo dos xx representa o *timestamp* e o dos yy o "oml_seq".

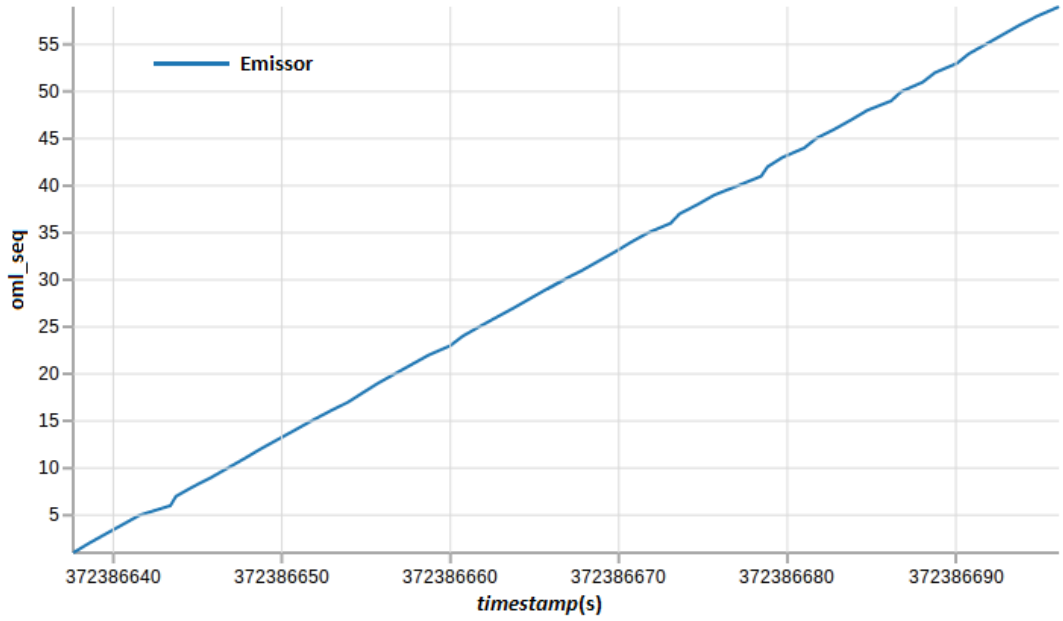


Figura 5.13: Gráfico do nó emissor no primeiro caso

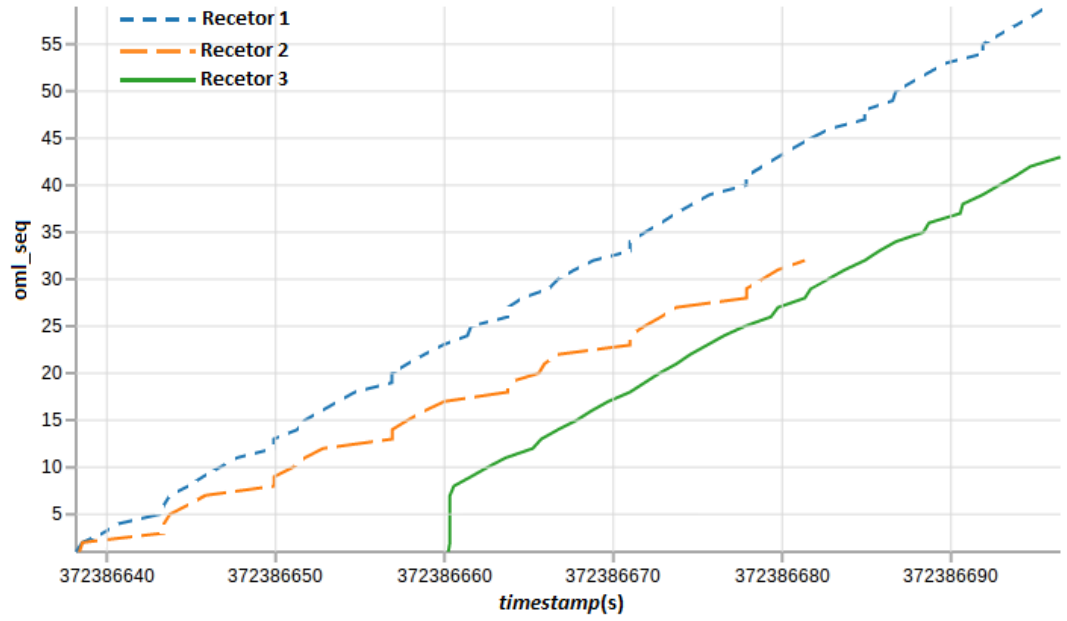


Figura 5.14: Gráfico dos três nós recetores no primeiro caso

Observando o gráfico dos recetores, conclui-se que a receção de pacotes não foi linear. O primeiro recetor recebeu todos os pacotes de forma mais ou menos linear, no entanto o segundo interropeu a receção antes do final da experiência, enquanto que o terceiro recetor só

iniciou a receção algum tempo depois do início desta experiência. A taxa de perda de pacotes foi de 24,9% no total.

A experiência foi repetida para uma taxa de transmissão de 2048 bps, da mesma forma que anteriormente, e podem ser observados os gráficos do nó que envia o tráfego na figura 5.15 e dos recetores na figura 5.16. No gráfico em 5.16 cada linha representa um dos nós, enquanto que no 5.15 apenas existe uma linha que representa o nó emissor.

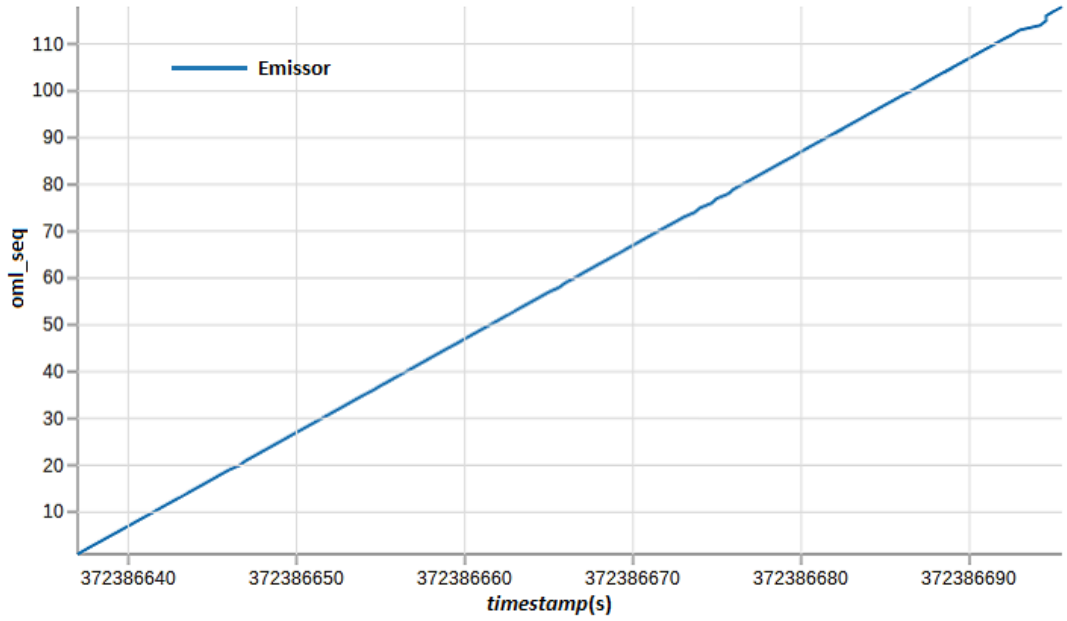


Figura 5.15: Gráfico do nó emissor no segundo caso

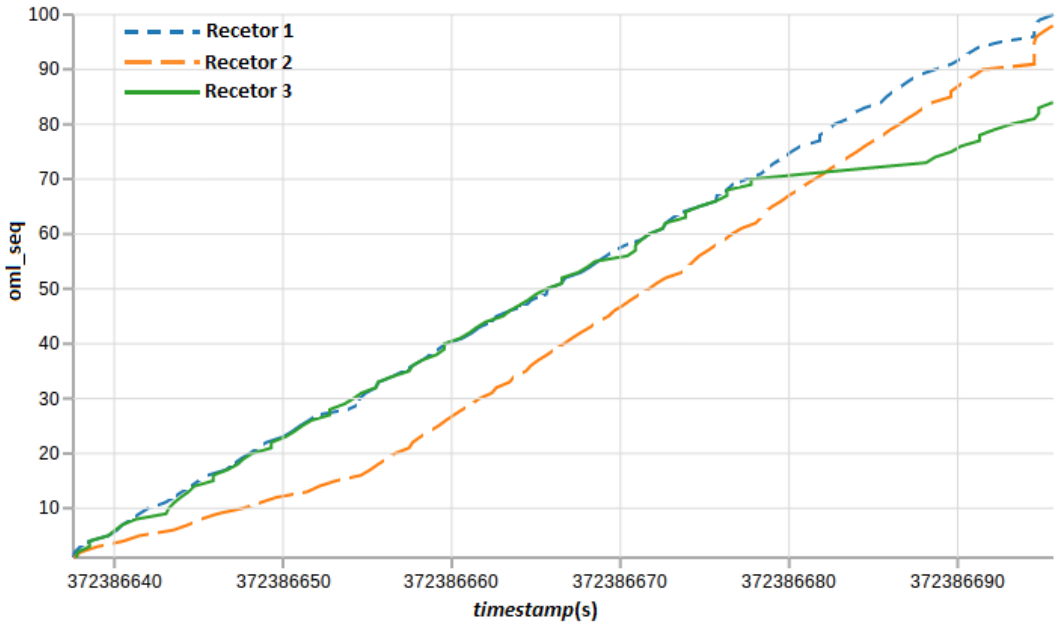


Figura 5.16: Gráfico dos três nós recetores no segundo caso

Neste caso as 3 transmissões estão mais próximas do que no caso anterior, mas mesmo assim com diferenças significativas, sendo que todos os nós recetores iniciaram e terminaram a receção dos pacotes nos mesmos instantes. A taxa de perda de pacotes ficou pelos 20.34% no total, o que é um valor inferior ao obtido no caso analisado anteriormente. O elevado valor da taxa de perda de pacotes deve-se, provavelmente, ao facto de não ser utilizado o GPS, como aconteceu no caso com múltiplos emissores de tráfego. O facto de nestas experiências ser utilizado tráfedo UDP impede a retransmissão de pacotes, o que permite obter uma maior confiança relativamente ao motivo destes valores da taxa de perda de pacotes. Estes pacotes nunca chegaram ao seu destino no momento em que as interfaces *wireless* não se encontravam no mesmo canal no nó emissor e no recetor, nem foram retransmitidos num momento seguinte em que já se encontravam no mesmo.

Pelos resultados obtidos nestas experiências verifica-se que o sistema de gestão é capaz de realizar experiências de forma eficiente com a interface WAVE e obter os resultados das mesmas.

5.1.3 Experiências Concorrentes

Nesta subsecção são apresentados os resultados de ativação de várias experiências de forma concorrente. Foi realizada uma experiência idêntica às anteriores, com transmissão

de dados, mas desta vez com seis nós, enviando tráfego dois a dois em simultâneo, ou seja, três experiências a correr paralelamente, em ambiente de laboratório. Foram configurados 3 *providers* e 3 *users*. Para não haver colisões, os serviços de *provider* foram configurados em canais diferentes. Foi criada uma interface em PHP que mostra as experiências que estão a correr nos nós num dado momento. Para auxiliar esta interface foi criada uma base de dados em MySQL, onde são guardados alguns dados das experiências, que são inseridos na ED quando a experiência é iniciada e removidos quando esta termina. Neste teste, o nó com o HRN "omf.my.drivein81" transmitiu para o "omf.my.drivein82", o "omf.my.drivein83" para o "omf.my.drivein84" e o "omf.my.drivein85" para o "omf.my.drivein90". Na figura 5.17 é ilustrada a interface referida, mostrando que estão três experiências a correr em simultâneo.

Figura 5.17: Experiências em execução nos nós

Nas figuras 5.18, 5.19 e 5.20 podem-se observar as tabelas utilizadas para gerar os gráficos apresentados mais adiante, sendo estes dados obtidos a partir das bases de dados criadas durante o decorrer das experiências, que contêm os dados das medições realizadas pelas mesmas. Nestas experiências, as medições foram enviadas para o servidor apenas a cada três recolhas de medições nas placas, configurando o "samples" com o valor 3. Na coluna "src_host" são apresentados os endereços IP dos nós de origem dos pacotes, cujo último octeto representa o número atribuído à respetiva placa. Na coluna "oml_seq" apresentam-se os *sequence numbers* atribuídos pelo OML aos pacotes, aquando a sua receção no nó, e na "oml_ts_server" apresentam-se os *timestamps* registados pelo OML no servidor. Como se pode verificar, cada um dos nós recetores apenas recebeu tráfego do seu emissor.

Table otr2_udp_in

Show 10 entries Search:

src_host	oml_seq	oml_ts_server
20.0.0.81	1	372386313.977845
20.0.0.81	2	372386317.47029
20.0.0.81	3	372386321.389725
20.0.0.81	4	372386323.245097
20.0.0.81	5	372386326.108453
20.0.0.81	6	372386329.212172
20.0.0.81	7	372386332.228888
20.0.0.81	8	372386335.124657
20.0.0.81	9	372386338.244355
20.0.0.81	10	372386341.37857

Showing 1 to 10 of 19 entries First Previous 1 2 Next Last

Figura 5.18: Dados recebidos na placa "omf.my.drivein82"

Table otr2_udp_in

Show 10 entries Search:

src_host	oml_seq	oml_ts_server
20.0.0.83	1	372386541.431548
20.0.0.83	2	372386544.349504
20.0.0.83	3	372386548.623714
20.0.0.83	4	372386551.577271
20.0.0.83	5	372386555.617569
20.0.0.83	6	372386558.588266
20.0.0.83	7	372386562.651625
20.0.0.83	8	372386565.592019
20.0.0.83	9	372386569.629699
20.0.0.83	10	372386572.652187

Showing 1 to 10 of 15 entries First Previous 1 2 Next Last

Figura 5.19: Dados recebidos na placa "omf.my.drivein84"

Table otr2_udp_in

Show 10 entries Search:

src_host	oml_seq	oml_ts_server
20.0.0.85	1	1003538643.81771
20.0.0.85	2	1003538656.10262
20.0.0.85	3	1003538663.07645
20.0.0.85	4	1003538666.08193
20.0.0.85	5	1003538672.05155
20.0.0.85	6	1003538678.78019
20.0.0.85	7	1003538686.11091
20.0.0.85	8	1003538699.15564

Showing 1 to 8 of 8 entries First Previous 1 Next Last

Figura 5.20: Dados recebidos na placa "omf.my.drivein90"

Nas figuras 5.21, 5.22 e 5.23 podem ser obervadas as transmissões nos nós recetores, gerados a partir dos dados das tabelas anteriormente apresentadas.

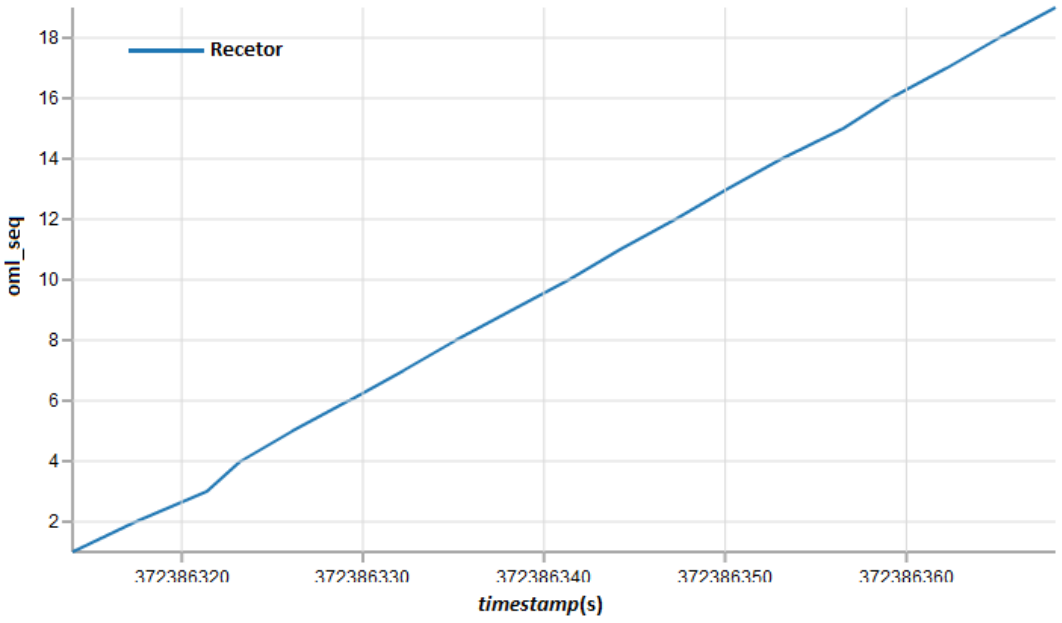


Figura 5.21: Dados recebidos na placa "omf.my.drivein82" em experiências concorrentes

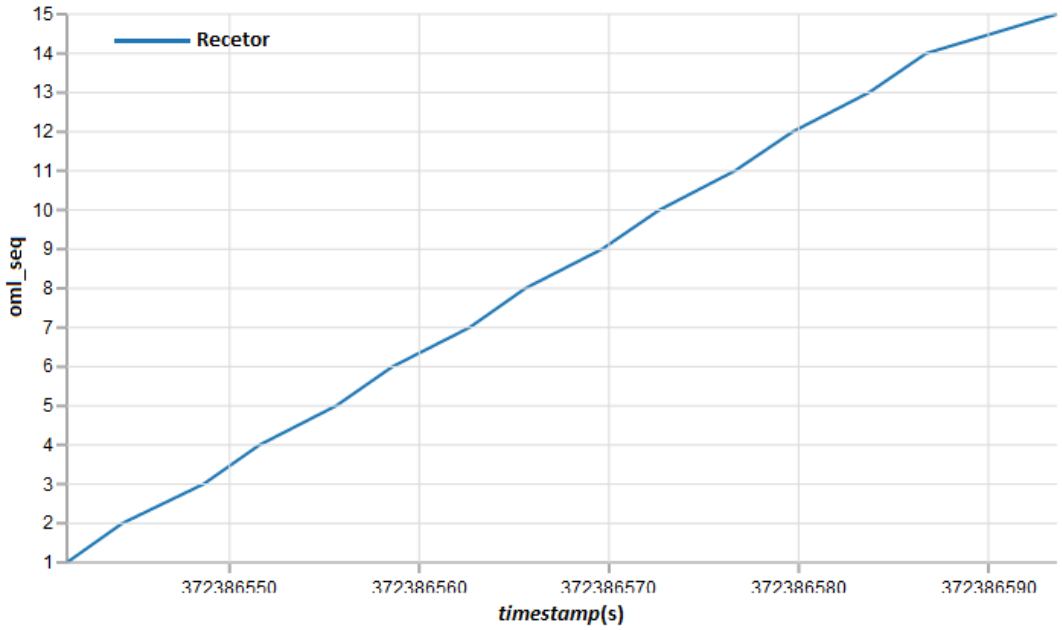


Figura 5.22: Dados recebidos na placa "omf.my.drivein84" em experiências concorrentes

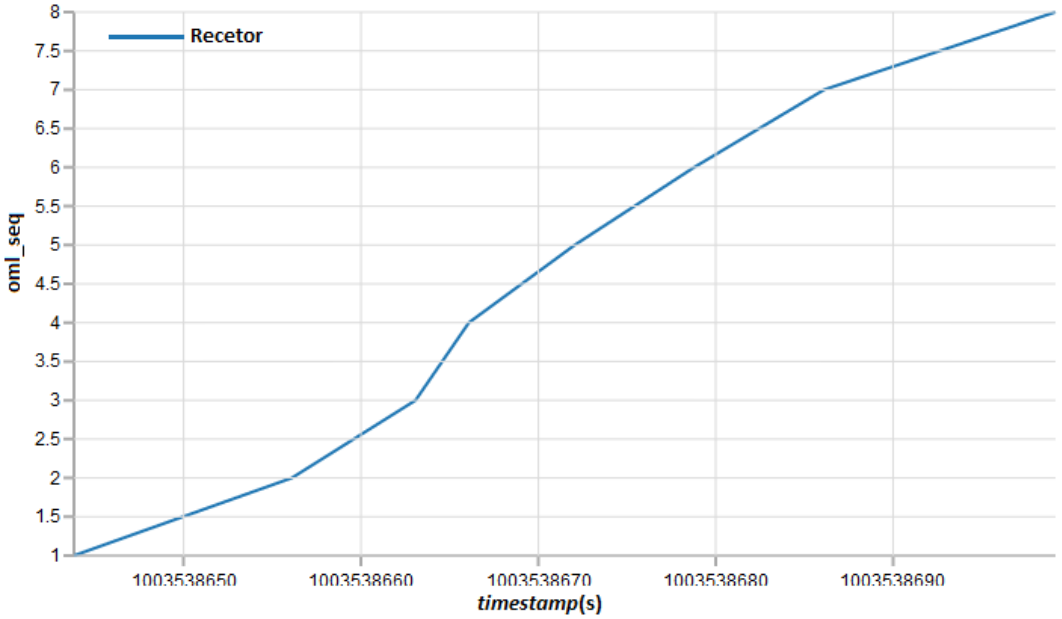


Figura 5.23: Dados recebidos na placa "omf.my.drivein90" em experiências concorrentes

No caso a experiência do gráfico 5.21 houve uma taxa de perda de pacotes de 0%, no caso da experiência do gráfico 5.22 a taxa foi de 21,05% e no caso da experiência do gráfico 5.23 esta taxa foi de 57,89%. Da mesma forma que ocorreu no caso da secção anterior não foi utilizado

GPS, havendo também uma dessincronização dos canais da interface *wireless* entre os nós. Nestas experiências, como os dados só foram recolhidos a cada três medições, obtiveram-se as informações de uma quantidade inferior de pacotes, explicando a grande diferença entre os valores da taxa de perda de pacotes. Pode-se então concluir que é possível executar múltiplas experiências em simultâneo na *testbed*, sem que haja problemas com a execução e recolha de resultados das mesmas.

5.2 Resultados Obtidos em Ambientes Reais

Para testar o funcionamento do sistema de gestão na execução e recolha de resultados de experiências em ambientes reais, foi feita uma experiência semelhante às da secção anterior, com envio e receção de pacotes UDP, usando a norma IEEE 802.11p para o efeito. Toda a configuração foi feita de forma semelhante para dois veículos, um a transmitir pacotes gerados pela aplicação OTG2, e outro a recebê-los com a aplicação OTR2. Foi ainda acrescentada a recolha das posições em que estes veículos se deslocaram na cidade do Porto, usando a aplicação GPS-Logger. Desta vez, a experiência correu sem tempo limite, até se obterem os resultados. Nas figuras 5.24 e 5.25 podem ser observadas as primeiras 10 posições reportadas pelos veículos aquando do início da receção dos pacotes, sendo o veículo 1 o emissor e o veículo 2 o recetor. O veículo 1 deslocava-se de Este para Oeste, e o veículo 2 deslocava-se de Oeste para Este. Os resultados apresentados nas figuras referidas foram obtidos com o uso da API do Google Maps [92].

Figura 5.24: Posições do veículo 1 no mapa

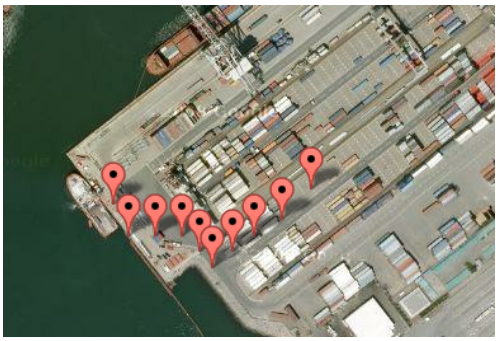


Figura 5.25: Posições do veículo 2 no mapa

A figura 5.26 apresenta os resultados de receção de tráfego no veículo 2. Pelos resultados obtidos verifica-se que esta receção não foi totalmente linear, apesar de nesta situação os nós já estarem a usar o GPS, o que permite uma sincronização dos canais por parte do *driver* da

interface da norma IEEE 802.11p. Isto deve-se aos fatores de ordem física, como obstáculos na estrada ou a distância entre os veículos. Como se pode observar, em alguns momentos a transmissão tornou-se linear, como entre os instantes marcados com os *timestamps* 372388000 e 372388500 (em segundos), o que corresponde às alturas em que os veículos se cruzaram e transmitiram informação entre si.

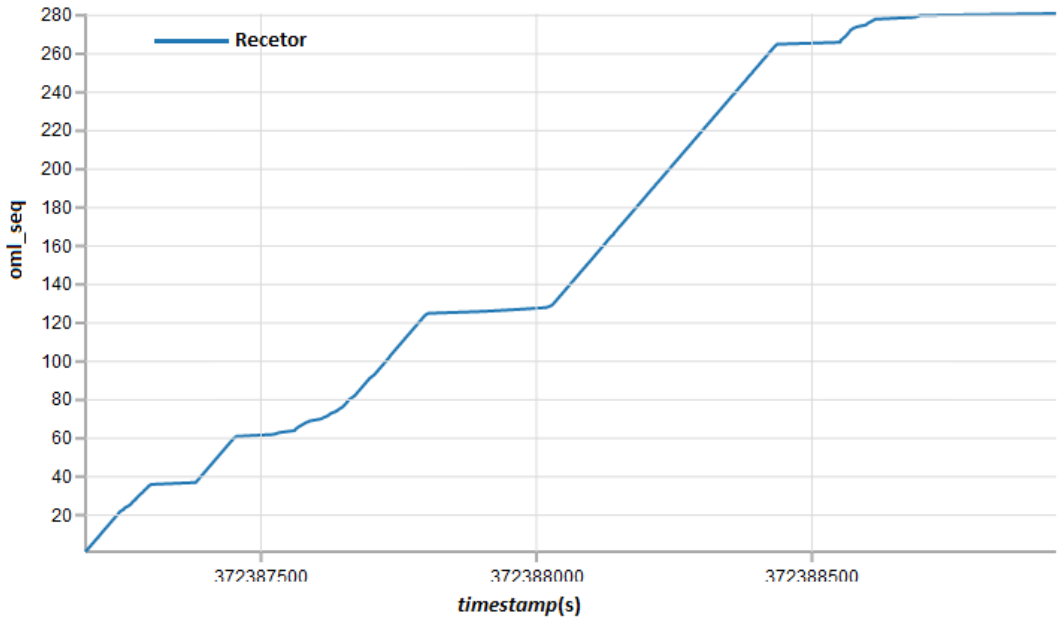


Figura 5.26: Dados recebidos no veículo 2 nas experiências em ambiente real

5.3 Conclusão

Ao longo deste capítulo puderam ser observados os resultados de experiências realizadas com o sistema de gestão. Foram executados testes com o envio de imagens de disco para um nó, para dois nós em simultâneo e para quatro nós em simultâneo. De seguida, procedeu-se à realização de testes de testes de escalonamento e controlo de experiências, com o envio de tráfego UDP entre os vários nós e a recolha dos dados das medições resultantes dos geradores e recetores de pacotes que foram utilizados para o envio e receção deste tráfego, respetivamente. Estes testes foram efetuados com recurso a uma placa a enviar em *broadcast* para três outras e a três placas a enviar para uma. Foram ainda realizados testes com a execução de experiências em simultâneo no sistema de gestão.

Os resultados obtidos com o envio das images de disco demonstram que este decorre em tempos aceitáveis, apesar de existir um grande aumento deste período quando o envio é feito

para quatro placas em simultâneo. No entanto, foi verificado que a qualidade de serviço da ligação 3G diminui quando existe uma grande utilização de largura de banda com esta quantidade de placas, devendo-se, provavelmente, ao facto de estarem todas localizadas na mesma célula da rede celular 3G. Verificou-se também que os nós, após obterem a imagem de disco, obtiveram as configurações próprias da imagem atual para a nova partição e reiniciaram na nova imagem, apresentando a informação respetiva na base de dados do OMF.

Nos testes de escalonamento e controlo de experiências de envio de tráfego, verificou-se que existe comunicação entre os vários nós pela interface da norma IEEE 802.11p, como era pretendido. No entanto, existiriam valores elevados de perda de pacotes, mesmo estando a usar nós fixos em ambiente de laboratório. Isto deve-se, provavelmente, ao facto de não se estar a usar o GPS durante o decorrer destas experiências, que funcionaria com o *driver* da norma IEEE 802.11p para garantir uma sincronização dos canais de controlo e de serviço. Desta forma, os pacotes só chegaram ao seu destino quando os canais se encontravam sincronizados de forma aleatória.

No caso de três experiências a correr em simultâneo, verificou-se que os pacotes foram recebidos nos nós de *user*, vindos dos respetivos *providers*, sem interferências entre eles.

Por fim, na experiência realizada num ambiente real, pôde-se obsevar a comunicação entre os veículo a ser realizada, bem como as suas coordenadas geográficas e os resultados das medições do envio de pacotes a serem enviados para a base de dados localizada no servidor do IT, o que indica que este sistema de gestão fornece a capacidade de controlar as experiências e as ligações no sistema real da rede veicular do DRIVE-IN.

Capítulo 6

Conclusão e Trabalho Futuro

6.1 Conclusão

Este trabalho de Dissertação consistiu na implementação de um sistema de gestão numa *testbed* de uma rede veicular, mais especificamente, do DRIVE-IN. Deste modo, após uma análise do estado de arte das redes veiculares e de alguns dos sistemas de gestão de *testbed* que existem, o trabalho pôde ser contextualizado e foi então escolhido um sistema de gestão apropriado para a *testbed*. No entanto, a nível de sistemas de gestão, não existe nenhum especificamente pensado para a gestão de *testbeds* de redes veiculares. O OMF foi o escolhido como o mais apropriado para a sua adaptação e instalação na *testbed*, desta forma surgiram alguns desafios que foram ultrapassados:

- Tornou-se impossível utilizar ligações cabladas nos carros, o que impediu uma comunicação entre os nós e com o sistema de gestão. Este problema foi ultrapassado com a instalação de *modems* de banda larga móvel e sua configuração no OMF;
- Como o OMF utiliza o protocolo PXE para fazer o carregamento de imagens, que depende das ligações cabladas, foi necessário encontrar uma solução para a atualização do *software* nos nós. O desafio foi respondido com uma alteração ao funcionamento nativo do OMF, de forma a enviar as imagens pela Internet. Para isto, foi utilizada uma ligação SSH feita ao nó que, de seguida, faz o *download* da imagem de disco e a coloca numa segunda partição, evitando problemas na partição principal, em caso de falha;
- As imagens de disco deveriam ter um tamanho sustentável, pois o seu envio pela rede 3G está mais sujeito a falhas e é mais demorado. Desta forma, foram geradas novas

imagens de disco com a ferramenta *Buildroot*, o uso de *uClibc*, e a instalação apenas das aplicações necessárias na *testbed*, bem como uma configuração mínima do *kernel* de Linux;

- A possibilidade de falhas nas ligações à rede celular e/ou bloqueios nas placas dos nós poderia causar o impedimento das comunicações permanentemente com os nós da *testbed* nos veículos. Isto seria um problema ainda mais acentuado com o facto de não se ter um acesso fácil aos veículos. Esta situação foi resolvida com a utilização do *Watchdog*, que foi configurado para reiniciar as placas sempre que existe uma falha de ligação ao servidor localizado no IT de Aveiro, ou sempre que a placa bloqueia completamente. Também foi implementado um serviço que permite a atualização automática do endereço IP dos nós na base de dados do OMF, visto este endereço ser dinâmico. Desta forma, foi conseguida uma forma de recuperação em caso de falha;

- A rede de experiências que o sistema de gestão deveria utilizar seria da norma IEEE 802.11p, no entanto o OMF não lhe oferece suporte, assim como qualquer outra opção para sistema de gestão. Portanto, o seu suporte foi extendido, incluindo a configuração dos serviços desta rede dinamicamente, durante a execução de experiências na *testbed*;

- A ligação à Internet nos veículos da *testbed*, pela rede 3G, deveria ser partilhada para usufruto dos motoristas e clientes dos mesmos. Após esta partilha ser efetuada, com a instalação de um serviço de AP nos nós, foi implementada uma forma de gerir a informação das ligações feitas a esta rede, enviando esta informação para o servidor do IT de Aveiro. Esta funcionalidade foi conseguida com o auxílio das ferramentas do sistema de gestão, o que possibilitou uma implementação simples de uma aplicação para recolher os dados dos *logs* gerados nos nós.

A implementação do sistema de gestão foi testada em várias situações possíveis, de forma a assegurar a sua viabilidade:

- Foram feitos testes com o envio de imagens de disco para um nó e para vários em simultâneo. Os resultados obtidos mostram que o envio das imagens de disco é feito num tempo aceitável, apesar de, no último caso, estes tempos terem subido significativamente. No entanto, conforme foi testado, verificou-se uma perda da qualidade de serviço sempre que era utilizada muita largura de banda em várias placas 3G ao mesmo tempo. Este problema poderá dever-se à utilização da mesma célula da rede celular durante a realização das experiências;

- Foram realizados testes de controlo de experiências com comunicações entre os nós com a norma IEEE 802.11p utilizando o sistema de gestão. Para tal, foi gerado tráfego UDP e enviado entre os nós, utilizando um gerador de tráfego propriamente instrumentado para o seu uso com o OMF. Foram criadas experiências que configuram serviços de *provider* e de *user* na interface IEEE 802.11p, possibilitando as comunicações pela mesma. Os testes realizados foram de envio de tráfego de um dos nós para três outros em *broadcast* e de envio de três nós para um único. Os resultados obtidos demonstram que, de facto, existe comunicação entre os nós, viabilizando o uso do sistema de gestão para o efeito de experiências com este tipo de interface.

- Foi testada a execução de múltiplas experiências em simultâneo na *testbed*, tendo-se obtido resultados idênticos aos do ponto anterior, o que mostra que é possível serem feitas várias experiências em simultâneo na *testbed* usando este sistema de gestão. No entanto, não está implementado um sistema de reversas na plataforma, o que impede de se conhecerem à partida os nós que já estão a ser utilizados em experiências.

- Por fim, os testes de envio de tráfego entre os nós da *testbed* foram realizados em ambiente real, estando estes nós instalados em veículos na cidade do Porto. Verificou-se uma comunicação entre estes veículos através da norma IEEE 802.11p, obtendo-se os resultados das medições com o envio e a receção de pacotes no servidor localizado no IT de Aveiro.

Após esta análise, verificou-se que o trabalho realizado e os resultados obtidos permitem concluir que é possível e viável a utilização deste sistema de gestão numa rede veicular como a do DRIVE-IN. Este sistema permite efetuar o envio de imagens de disco em tempos aceitáveis para os diferentes nós e ter resultados obtidos de dados de medições de experiências realizadas na *testbed*. Também se obtiveram as informações que se pretenderam dos nós da *testbed* no servidor local, bem como algum controlo sobre eles. Pode-se afirmar que, de uma maneira geral, os objetivos deste trabalho foram todos atingidos.

6.2 Trabalho Futuro

Existe ainda algum trabalho a ser efetuado no que respeita ao sistema de gestão. Pretende-se melhorar a sua interação com os utilizadores, fornecendo-lhes meios mais simples de visualizarem os recursos da *testbed* do DRIVE-IN e de controlarem as experiências e envio de *software* remotamente.

Existem também experiências de larga escala a ser efetuadas num futuro próximo, usando o sistema de gestão para seu auxílio. Com estas experiências pretende-se estudar o desempenho da tecnologia das VANET na cidade do Porto. Para tal, vai ser em breve implementado o sistema de gestão em unidades de bordo em táxis da respetiva cidade.

A gestão das comunicações e utilizadores também será estendida para se obter a informação dos eventos gerados pelas interfaces da norma IEEE 802.11p, juntamente com a localização em que o evento foi despoletado. Desta forma, será possível ter uma informação das comunicações veiculares num ambiente real de uma forma simples e direta.

Bibliografia

[1] DRIVE-IN. [online]. http://drive-in.cmuportugal.org/, Maio 2012.

[2] European newsletter published article about the DRIVE-In project. [online]. http://www.cmuportugal.org/tiercontent.aspx?id=3434, Maio 2012.

[3] NICTA - service delivery and testbed framework. [online]. http://www.nicta.com.au/research/projects/tempo/, Maio 2012.

[4] F. Neves, A. Cardote, R. Moreira, and S. Sargento. Real-world evaluation of ieee 802.11 p for vehicular networks. In *Proceedings of the Eighth ACM international workshop on Vehicular inter-networking*, pages 89–90. ACM, 2011.

[5] R.S. Alves, I.V. Campbell, R.S. Couto, M.E.M. Campista, I.M. Moraes, M.G. Rubinstein, L.H.M.K. Costa, O.C.M.B. Duarte, and M. Abdalla. Redes veiculares: Princıpios, aplicações e desafios. *Minicursos do Simpósio Brasileiro de Redes de Computadores, SBRC*, 2009.

[6] H. Moustafa, S.M. Senouci, and M. Jerbi. Introduction to vehicular networks. *Zhang, Yan (Hrsg.): Vehicular networks: Techniques, Standards and Applications*, pages 1–20, CRC Press, 2009.

[7] The future of intelligent transport systems (ITS) ≪ engaged IT for the CIO. [online]. http://mubbisherahmed.wordpress.com/2011/11/29/the-future-of-intelligent-transport-systems-its/, Maio 2012.

[8] H. Moustafa and Y. Zhang. *Vehicular networks: techniques, standards, and applications*. CRC Press, 2009.

[9] R. Baumann. Vehicular ad hoc networks (vanet). *Ad Hoc Networks*, 2004.

[10] T. Kosch, C. Schwingenschlogl, and L. Ai. Information dissemination in multihop inter-vehicle networks. In *Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference on*, pages 685–690. IEEE, 2002.

[11] I. Chlamtac, M. Conti, and J.J.N. Liu. Mobile ad hoc networking: imperatives and challenges. *Ad Hoc Networks*, 1(1):13–64, Elsevier Science, 2010.

[12] ITS standards fact sheets - U.S. department of transportation. [online]. http://www.standards.its.dot.gov/fact_sheet.asp?f=80, Maio 2012.

[13] Y. Qian and N. Moayeri. Medium access control protocols for vehicular networks. *Vehicular Networks: Techniques, Standards, and Applications*, pages 41–62, CRC Press, 2009.

[14] PCQuest : Technology : Intelligent transportation using VANET. [online]. http://pcquest.ciol.com/content/technology/2009/109020101.asp, Maio 2012.

[15] Vanet's - vehicular adhoc networks. [online]. http://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2010_2/lemos/introducao.html, Maio 2012.

[16] Car 2 car communication consortium. [online]. http://www.car-to-car.org/, Maio 2012.

[17] M. Nekovee. Sensor networks on the road: the promises and challenges of vehicular ad hoc networks and grids. In *Workshop on Ubiquitous Computing and e-Research, Edinburgh, UK*, 2005.

[18] J.J. Blum, A. Eskandarian, and L.J. Hoffman. Challenges of intervehicle ad hoc networks. *Intelligent Transportation Systems, IEEE Transactions on*, 5(4):347–351, IEEE, 2004.

[19] M. Kihl. Vehicular network applications and services. *Vehicular Networks: Techniques, Standards, and Applications*, pages 21–40, CRC Press, 2009.

[20] W. Kremer. Realistic simulation of a broadcast protocol for an inter vehicle communication system (ivcs). In *Vehicular Technology Conference, 1991. Gateway to the Future Technology in Motion., 41st IEEE*, pages 624–629. IEEE, 1991.

[21] O.K. Tonguz and G. Ferrari. *Ad hoc wireless networks*. Wiley Online Library, 2005.

[22] O.K. Tonguz, N. Wisitpongphan, J.S. Parikh, F. Bai, P. Mudalige, and V.K. Sadekar. On the broadcast storm problem in ad hoc wireless networks. In *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, pages 1–11. IEEE, 2006.

[23] M. Torrent-Moreno, D. Jiang, and H. Hartenstein. Broadcast reception rates and effects of priority access in 802.11-based vehicular ad-hoc networks. In *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, pages 10–18. ACM, 2004.

[24] IETF tools - autoconf. http://tools.ietf.org/wg/autoconf/, Maio 2012.

[25] ISO - technical committees - TC 204 - intelligent transport systems. [online]. http://www.iso.org/iso/iso_technical_committee?commid=54706, Maio 2012.

[26] ETSI intelligent transport systems. [online]. http://www.etsi.org/website/technologies/intelligenttransportsystems.aspx, Maio 2012.

[27] Audi worldwide. [online]. http://www.audi.com/, Maio 2012.

[28] BMW automobiles - website of the BMW AG. [online]. http://www.bmw.com/, Maio 2012.

[29] Official site of chrysler cars, convertibles and minivans | chrysler. [online]. http://www.chrysler.com/, Maio 2012.

[30] Fiat. [online]. http://www.fiat.com/, Maio 2012.

[31] Renault.com - car manufacturer renault's official international website - renault. [online]. http://www.renault.com/, Maio 2012.

[32] Volkswagen international. [online]. http://www.volkswagen.com/, Maio 2012.

[33] Automotive solutions | connected services | windows embedded automotive 7. [online]. http://www.microsoft.com/windowsembedded/en-us/evaluate/windows-embedded-automotive-7.aspx, Maio 2012.

[34] S. Biswas, R. Tatchikou, and F. Dion. Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety. *Communications Magazine, IEEE*, 44(1):74–82, IEEE, 2006.

[35] A. Benslimane. Optimized dissemination of alarm messages in vehicular ad-hoc networks (vanet). *High Speed Networks and Multimedia Communications*, pages 655–666, Springer, 2004.

[36] C. Maihofer, C. Cseh, W. Franz, and R. Eberhardt. Performance evaluation of stored geocast. In *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, volume 5, pages 2901–2905. IEEE, 2003.

[37] C. Maihofer. A survey of geocast routing protocols. *Communications Surveys & Tutorials, IEEE*, 6(2):32–42, IEEE, 2004.

[38] H. Krishnan, F. Bai, and G. Holland. Commercial and public use applications. *Vehicular Networking*, pages 1–28, 2010. Wiley Online Library.

[39] V. Cristea, V. Gradinescu, C. Gorgorin, R. Diaconescu, and L. Iftode. Simulation of vanet applications. *Automotive Informatics and Communicative Systems*, Information Science Reference. 2009.

[40] Michel Ferreira, Ricardo Fernandes, Hugo Conceição, Wantanee Viriyasitavat, and Ozan K. Tonguz. Self-organized traffic control. *VANET10, Chicago, Illinois, USA*, pages 85–89, 2010.

[41] A. Stevanovic. *Adaptive traffic control systems: domestic and foreign state of practice.* Number Projeto 20-5 (Tópico 40-03). 2010.

[42] J. Bronsted and L.M. Kristensen. Specification and performance evaluation of two zone dissemination protocols for vehicular ad-hoc networks. In *Proceedings of the 39th annual Symposium on Simulation*, pages 68–79. IEEE Computer Society, 2006.

[43] W. Chen and S. Cai. Ad hoc peer-to-peer network architecture for vehicle safety communications. *Communications Magazine, IEEE*, 43(4):100–107, IEEE, 2005.

[44] O.K. Tonguz and M. Boban. Multiplayer games over vehicular ad hoc networks: A new application. *Ad Hoc Networks*, 8(5):531–543, Elsevier, 2010.

[45] C. Olaverri-Monreal, P. Gomes, R. Fernandes, F. Vieira, and M. Ferreira. The see-through system: A vanet-enabled assistant for overtaking maneuvers. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 123–128. IEEE, 2010.

[46] ns-3. [online]. http://www.nsnam.org/, Junho 2012.

[47] OMNeT++ network simulation framework. [online]. http://www.omnetpp.org/, Junho 2012.

[48] Emulab - network emulation testbed. [online]. http://www.emulab.net/, Junho 2012.

[49] T. Rakotoarivelo, G. Jourjon, M. Ott, and I. Seskar. Omf: a control and management framework for networking testbeds. *Operating systems review*, 43(4):54, 2009.

[50] OMF: control and management framework. [online]. http://omf.mytestbed.net/projects/omf, Junho 2012.

[51] M. Mastrogiovanni, A. Modesti, and C. Petrioli. James: Java test-bed management system. In *Vehicular Technology Conference, 2009. VTC Spring 2009. IEEE 69th*, pages 1–6. IEEE, 2009.

[52] Java. [online]. http://www.java.com/en/, Junho 2012.

[53] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: A wireless sensor network testbed. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 68. IEEE Press, 2005.

[54] MySQL : open source database. [online]. http://www.mysql.com/, Junho 2012.

[55] PHP: hypertext preprocessor. [online]. http://www.php.net/, Junho 2012.

[56] The perl programming language. [online]. http://www.perl.org/, Junho 2012.

[57] P. Hurni, M. Anwander, G. Wagenknecht, T. Staub, and T. Braun. Tarwis: a test-bed management architecture for wireless sensor network testbeds. In *Proceedings of the 7th International Conference on Network and Services Management*, pages 320–323. International Federation for Information Processing, 2011.

[58] Shibboleth. [online]. http://shibboleth.net/, Junho 2012.

[59] Debian – the universal operating system. [online]. http://www.debian.org/, Junho 2012.

[60] Python programming language. [online]. http://www.python.org/, Junho 2012.

[61] DES-Testbed. [online]. http://www.des-testbed.net/, Junho 2012.

[62] SNMP. [online]. http://www.snmp.com/, Junho 2012.

[63] SNMP4J - free open source SNMP API for java. [online]. http://www.snmp4j.org/, Junho 2012.

[64] JavaView. [online]. http://www.javaview.de/, Junho 2012.

[65] Orbit. [online]. http://www.orbit-lab.org/, Junho 2012.

[66] WINLAB. [online]. http://www.winlab.rutgers.edu/, Junho 2012.

[67] NICTA. [online]. http://www.nicta.com.au/, Junho 2012.

[68] Ruby programming language. [online]. http://www.ruby-lang.org/en/, Junho 2012.

[69] O. Hahm, M. Gunes, F. Juraschek, B. Blywis, and N. Schmittberger. An experimental facility for wireless multi-hop networks in future internet scenarios. In *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pages 48–57. IEEE, 2011.

[70] SQLite. [online]. http://www.sqlite.org/, Junho 2012.

[71] OML - measurement library - OMF developer portal. [online]. http://mytestbed.net/projects/oml, Junho 2012.

[72] PXE Intel. Preboot execution environment (pxe) specification. *Intel Corporation*, 1999.

[73] P. Cowan. What is pxe?, 2008.

[74] XMPP technologies: PubSub - the XMPP standards foundation. [online]. http://xmpp.org/about-xmpp/technology-overview/pubsub/, Junho 2012.

[75] C. Ameixieira, J. Matos, R. Moreira, A. Cardote, A. Oliveira, and S. Sargento. An ieee 802.11 p/wave implementation with synchronous channel switching for seamless dual-channel access (poster). In *Vehicular Networking Conference (VNC), 2011 IEEE*, pages 214–221. IEEE, 2011.

[76] TP-LINK. [online]. http://www.tp-link.us/, Junho 2012.

[77] ZWAME fórum. [online]. http://forum.zwame.pt/, Junho 2012.

[78] Ubuntu. http://www.ubuntu.com/, Junho 2012.

[79] Ignite realtime: Openfire server. [online]. http://www.igniterealtime.org/projects/openfire/, Junho 2012.

[80] phpMyAdmin. http://www.phpmyadmin.net/, Junho 2012.

[81] The apache HTTP server project. [online]. http://httpd.apache.org/, Junho 2012.

[82] Git. [online]. http://git-scm.com/, Junho 2012.

[83] Draisberghof - USB_ModeSwitch. [online]. http://www.draisberghof.de/usb_modeswitch/, Junho 2012.

[84] Connect to internet using a huawei mobile broadband modem - bifferboard. [online]. https://sites.google.com/site/bifferboard/Home/howto/connect-to-internet-using-a-huawei-mobile-broadband-modem, Junho 2012.

[85] net-ssh. [online]. http://rubygems.org/gems/net-ssh, Junho 2012.

[86] DSL. [online]. http://www.damnsmalllinux.org/, Junho 2012.

[87] Tiny core linux, micro core linux. [online]. http://distro.ibiblio.org/tinycorelinux/, Junho 2012.

[88] Buildroot. [online]. http://buildroot.uclibc.org/, Junho 2012.

[89] uClibc. [online]. http://www.uclibc.org/, Junho 2012.

[90] BusyBox. [online]. http://www.busybox.net/, Junho 2012.

[91] hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS authenticator. [online]. http://hostap.epitest.fi/hostapd/, Junho 2012.

[92] Google maps. [online]. https://maps.google.com/, Junho 2012.